

Reference

NAT'L INST. OF STAND & TECH



A11106 261549

NBS
Publi-
cations

NBSIR 82-2619

Functional Specifications for A Federal Information Processing Standard Data Dictionary System

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Programming Science and Technology
Institute for Computer Sciences and Technology
Washington, DC 20234

January 1983



QC

100

,U56

82-2619

1983

DEPARTMENT OF COMMERCE

NATIONAL BUREAU OF STANDARDS

FEB 7 1983

not acc + ket

12-20

12-20

1983

NBSIR 82-2619

**FUNCTIONAL SPECIFICATIONS FOR A
FEDERAL INFORMATION PROCESSING
STANDARD DATA DICTIONARY SYSTEM**

Editors

Patricia A. Konig

Alan Goldfine

Judith J. Newton

U.S. DEPARTMENT OF COMMERCE

National Bureau of Standards

Center for Programming Science and Technology

Institute for Computer Sciences and Technology

Washington, DC 20234

January 1983

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

PART I
TABLE OF CONTENTS

	Page
PREFACE.....	viii
1. INTRODUCTION	2
1.1 Background	2
1.2 Expected Benefits	3
1.3 ICST Project Approach and Status	4
1.3.1 Project Approach	4
1.3.2 Project Status	5
1.4 Scope of Report	7
2. MANAGEMENT OVERVIEW	8
2.1 BACKGROUND	8
2.2 FUNCTIONAL SPECIFICATIONS FOR THE FIPS DDS	9
2.3 STRUCTURE OF THE DDS	10
2.3.1 The Dictionary	10
2.3.2 The Dictionary Schema	11
2.3.3 Customizing the DDS	14
2.4 THE SYSTEM STANDARD SCHEMA	14
2.4.1 System Standard Entity-Types	14
2.4.2 System Standard Relationship-Types	15
2.4.3 System Standard Attribute-Types	16
2.5 THE DDS COMMAND LANGUAGE	17
2.5.1 Interaction with the Dictionary Schema	17
2.5.2 Interaction with the Dictionary	18
2.6 THE DDS SOFTWARE INTERFACES	21
2.7 SECURITY FACILITIES OF THE FIPS DDS	22

PART II

TABLE OF CONTENTS

		PAGE
	ERRATA	ERRATA- 1
1.	INTRODUCTION.....	1- 1
2.	THE DDS ENVIRONMENT AND DDS DEFAULTS.....	2- 1
2.1	THE DDS ENVIRONMENT.....	2- 1
2.2	DDS DEFAULTS.....	2- 2
2.2.1	MODE FOR MAINTENANCE OF THE DICTIONARY SCHEMA....	2- 2
2.2.2	MODE FOR MAINTENANCE OF THE DICTIONARY.....	2- 3
2.2.3	DEFAULT STATUS.....	2- 3
2.2.4	VERSION DEFAULT.....	2- 4
2.2.5	CODE VALUES DEFAULT.....	2- 5
3.	THE DICTIONARY AND DICTIONARY SCHEMA AND THEIR STRUCTURE.....	3- 1
3.1	THE STRUCTURE OF THE DICTIONARY SCHEMA.....	3- 2
3.1.1	META-ENTITY-TYPES.....	3- 3
3.1.2	META-RELATIONSHIP-TYPES.....	3- 6
3.1.3	META-ATTRIBUTE-TYPES OF META-ENTITY- TYPES.....	3- 8

3.1.4	META-ATTRIBUTE-TYPES OF META-RELATION- SHIP-TYPES.....	3-	22
3.1.5	THE STATUS FACILITY OF THE DICTIONARY SCHEMA.....	3-	31
3.1.6	THE STAGE FACILITY OF THE DICTIONARY SCHEMA.....	3-	32
3.2	THE DICTIONARY SCHEMA.....	3-	33
3.3	CUSTOMIZATION OF THE DICTIONARY SCHEMA.....	3-	34
3.4	THE DICTIONARY.....	3-	35
4.	THE SYSTEM-STANDARD SCHEMA.....	4-	1
4.1	ENTITY-TYPES.....	4-	2
4.2	RELATIONSHIP-TYPES.....	4-	11A
4.3	RELATIONSHIP-CLASS-TYPES.....	4-	41
4.4	ATTRIBUTE-TYPES.....	4-	44
4.5	ATTRIBUTE-GROUP-TYPES.....	4-	58
4.6	ATTRIBUTE-TYPE-VALIDATION-PROCEDURE.....	4-	62
4.7	ATTRIBUTE-TYPE-VALIDATION-DATA.....	4-	63
4.8	STATUS-NAME.....	4-	63
4.9	STAGE-NAME.....	4-	65

4.10	ATTRIBUTE-TYPES AND ATTRIBUTE-GROUP-TYPES ASSOCIATED WITH ENTITY-TYPES.....	4-	65
4.11	ATTRIBUTE-TYPES AND ATTRIBUTE-GROUP-TYPES ASSOCIATED WITH RELATIONSHIP-TYPES.....	4-	74
5.	COMMANDS FOR INTERACTION WITH THE DICTIONARY SCHEMA.....	5-	1
5.1	SCHEMA MAINTENANCE COMMANDS.....	5-	1
5.1.1	ABOLISH-META-ENTITY COMMAND.....	5-	3
5.1.2	ABOLISH-META-ENTITY-WITH-LOCK COMMAND.....	5-	5
5.1.3	ABOLISH-META-RELATIONSHIP COMMAND.....	5-	5
5.1.4	ALTER-META-ENTITY COMMAND.....	5-	7
5.1.5	ALTER-META-ENTITY-WITH-LOCK COMMAND.....	5-	12
5.1.6	ALTER-META-RELATIONSHIP COMMAND.....	5-	12
5.1.7	CHANGE-META-NAME COMMAND.....	5-	14
5.1.8	CREATE-META-ENTITY COMMAND.....	5-	15
5.1.9	CREATE-META-RELATIONSHIP COMMAND.....	5-	21
5.1.10	REPLACE-META-RELATIONSHIP COMMAND.....	5-	24
5.2	SCHEMA REPORTING COMMANDS.....	5-	27
5.2.1	IS-META-RELATED-COMMAND.....	5-	27
5.2.2	META-CATALOG-COMMAND.....	5-	28
5.2.3	META-LIST-COMMAND.....	5-	29
5.2.4	META-TRACE COMMAND.....	5-	32
6.	COMMANDS FOR INTERACTION WITH THE DICTIONARY.....	6-	1
6.1	GENERAL RULES.....	6-	1
6.1.1	ENTITY-NAME RULES.....	6-	2
6.1.2	USE-AS-IDENTIFIER ATTRIBUTE-TYPE RULES.....	6-	3

6.1.3	ATTRIBUTE RULES.....	6-	5
6.1.4	TEXT ATTRIBUTE-TYPES.....	6-	8
6.1.5	RULES FOR USAGE-NAMES ATTRIBUTE-GROUPS.....	6-	10
6.1.6	AUDIT ATTRIBUTE-TYPES.....	6-	12A
6.1.7	NULL ATTRIBUTES.....	6-	12A
6.2	QUALIFICATION.....	6-	13
6.2.1	QUALIFY COMMAND.....	6-	16
6.2.1.1	ENTITY-SELECTION CLAUSE.....	6-	19
6.2.1.2	ALTERNATE-NAME-SELECTION CLAUSE.....	6-	21
6.2.1.3	PRIMARY-NAME-RESTRICTION CLAUSE.....	6-	24
6.2.1.4	RELATIONSHIP-RESTRICTION CLAUSE.....	6-	26
6.2.1.5	AUDIT-ATTRIBUTE-RESTRICTION CLAUSE.....	6-	30
6.2.1.6	ATTRIBUTE-RESTRICTION CLAUSE.....	6-	32
6.2.1.7	TEXT-STRING-RESTRICTION CLAUSE.....	6-	35
6.2.1.8	CLASSIFICATION-RESTRICTION CLAUSE.....	6-	36
6.2.2	UNION, INTERSECTION, SET-DIFFERENCE COMMANDS.....	6-	38
6.2.3	DISQUALIFY-LIST COMMAND.....	6-	40
6.2.4	SAVE-LIST COMMAND.....	6-	41
6.2.5	RUN COMMAND.....	6-	43
6.2.6	DROP-PROCEDURE COMMAND.....	6-	44
6.2.7	LIST-QUALIFICATIONS COMMAND.....	6-	45
6.3	MAINTENANCE COMMANDS.....	6-	46
6.3.1	ADD-ENTITY COMMAND.....	6-	46
6.3.2	ADD-RELATIONSHIP COMMAND.....	6-	51
6.3.3	CHANGE-STATUS COMMAND.....	6-	58
6.3.4	COPY COMMAND.....	6-	62
6.3.5	DECLARE COMMAND.....	6-	65
6.3.6	DELETE-ENTITY COMMAND.....	6-	69
6.3.7	DELETE-RELATIONSHIP COMMAND.....	6-	71
6.3.8	MODIFY-ENTITY COMMAND.....	6-	77
6.3.9	MODIFY-RELATIONSHIP COMMAND.....	6-	81
6.3.10	RENAME COMMAND.....	6-	84
6.3.11	RENUMBER COMMAND.....	6-	87

6.4	REPORT COMMANDS.....	6- 91
6.4.1	CATALOG COMMAND.....	6- 91
6.4.2	IMPACT-OF-CHANGE COMMAND.....	6- 98
6.4.3	IMPLICIT-ENTITIES COMMAND.....	6-105
6.4.4	LIST COMMAND.....	6-107
6.4.4A	ORPHANED-ENTITIES COMMAND.....	6-108
6.4.4B	PRODUCE-SYNTAX COMMAND.....	6-108B
6.4.5	VERSION-REPORT COMMAND.....	6-108E
6.5	QUERY COMMANDS.....	6-111
6.5.1	GENERAL QUERY COMMAND.....	6-111
6.5.2	SAVE-QUERY COMMAND.....	6-113
6.5.3	RUN-QUERY COMMAND.....	6-114
6.5.4	DELETE-QUERY COMMAND.....	6-115
6.5.5	LIST-QUERIES COMMAND.....	6-115
6.5.6	SPECIAL QUERY COMMANDS.....	6-116
6.5.6.1	IMPLICIT-ENTITY QUERY.....	6-116
6.5.6.2	ENTITY-AND-ATTRIBUTE QUERY.....	6-117
6.5.6.3	ALTERNATE-NAME-AND-CONTEXT QUERY.....	6-118
6.5.6.4	CONTAINS-STRING QUERY.....	6-119
6.5.6.5	RELATIONSHIP QUERY.....	6-120
6.5.6.6	AUDIT QUERY.....	6-121
6.5.6.7	CLASSIFICATION QUERY.....	6-122
7.	DDS SOFTWARE INTERFACES.....	7- 1
7.1	GENERATE-STRUCTURE-FOR-COBOL COMMAND.....	7- 1
7.2	THE EXPORT/IMPORT FACILITY.....	7- 9
7.2.1	INTEGRITY CONSIDERATIONS.....	7- 9
7.2.2	SCHEMA EQUIVALENCE AND THE ESSENTIAL SCHEMA.....	7- 11
7.2.2.1	EXTRACT-ESSENTIAL-SCHEMA COMMAND.....	7- 13
7.2.2.2	COMPARE-ESSENTIAL-SCHEMAS COMMAND.....	7- 15
7.2.3	EXPORT/IMPORT PROCEDURE.....	7- 16
7.2.4	EXPORT/IMPORT COMMANDS.....	7- 18
7.2.4.1	EXTRACT-SUBSET COMMAND.....	7- 19

7.2.4.2	CREATE-DICTIONARY COMMAND.....	7-	23
7.2.4.3	LOAD-DICTIONARY COMMAND.....	7-	25
7.2.4.4	IMPORT-SUBSET COMMAND.....	7-	26
7.3	DDS PROGRAM INTERFACE - THE CALL DDS COMMAND.....	7-	29
8.	DICTIONARY ADMINISTRATOR COMMANDS AND TOOLS.....	8-	1
8.1	THE DDS SECURITY FACILITY.....	8-	1
8.1.1	DICTIONARY-USER ENTITY-TYPE.....	8-	2
8.1.2	RULES FOR THE ENTITY-TYPE DICTIONARY-USER.....	8-	11
8.1.3	ACCESS-CONTROLLER ENTITY-TYPE.....	8-	12A
8.1.4	RULES FOR THE ENTITY-TYPE ACCESS-CONTROLLER.....	8-	14
8.1.5	EFFECTS OF THE DDS SECURITY FACILITY.....	8-	14
8.1.5.1	QUALIFICATION COMMANDS.....	8-	14
8.1.5.2	MAINTENANCE COMMANDS.....	8-	15
8.1.5.3	REPORT COMMANDS.....	8-	16
8.1.5.4	QUERY COMMANDS.....	8-	17
8.1.5.5	DDS SOFTWARE INTERFACE COMMANDS.....	8-	17
8.2	LOCAL SECURITY COMMANDS.....	8-	18
8.2.1	ADD-SECURITY COMMAND.....	8-	18
8.2.2	DELETE-SECURITY COMMAND.....	8-	20
8.2.3	MODIFY-SECURITY COMMAND.....	8-	21
8.2.4	ASSIGN-KEY COMMAND.....	8-	24
8.2.5	DELETE-KEY COMMAND.....	8-	27
8.3	DICTIONARY ADMINISTRATOR TOOLS.....	8-	29
8.3.1	TOOLS FOR DICTIONARY SET UP.....	8-	29
8.3.2	TOOLS FOR DICTIONARY CONTINUITY.....	8-	30
8.3.3	TOOLS FOR PERFORMANCE IMPROVEMENT.....	8-	31
	APPENDIX A.....	A-	1
	APPENDIX B.....	B-	1

PREFACE

We gratefully acknowledge the cooperation of the Federal users and developers of Data Dictionary Systems who assisted the Institute for Computer Science and Technology in identifying current and projected Federal requirements for data dictionary software. The names of the participating individuals are listed in alphabetical order.

Capt. Dennis Bruns
Charles W. Burlingame
Peter Cuomo
Phillip Dawson
Ruth Dyke
Mary Kay Daniels Ganning
Ann Glascock
Josi Hillary
Carolyn Jackson
R. Scott Jennings
Neil S. Jarrett, Jr.
Brenda King
Jeanne Kline
Belkis Leong-Hong
Robert Levi
Robert Lovelace
John Martin
Susan McFarland
Daniel Moreno

Linda Moyer
Cameron Murchison
Harvey Payne
James Radosevich
Ric Rawson
Nancy Reid
Lawrence Ries
Pat Robinson
Bruce Rosen
William J. Ross
William Selfridge
Major Charles Shidesky
Lt. Daniel Simes
Jerry Szablonsky
Lt. Bridgette Taylor
Catherine Ward
Sheila Watkins
Roxanne Williams
Margaret Wisner

Two members of the Data Management and Programming Languages Division read and criticized early versions of this report. They are Helen Wood, Division Chief, and Roy Saltman, Group Leader. Part II of this report was prepared for the Institute for Computer Sciences and Technology by Dr. Henry Lefkovits, Dr. Edgar Sibley, and Dr. Anthony J. Winkler of Alpha Omega Group, Inc. We urge readers to provide comments and to further interact with us on this document. Responses should be directed to the address below.

Mrs. Patricia A. Konig
Institute for Computer Sciences
and Technology
National Bureau of Standards
Technology Building, Room A265
Washington, D.C. 20234

FUNCTIONAL SPECIFICATIONS FOR A
FEDERAL INFORMATION PROCESSING STANDARD
DATA DICTIONARY SYSTEM

Editors

Patricia A. Konig
Alan Goldfine
Judith J. Newton

This interim report contains Functional Specifications for the basic functions that data dictionary software must perform to satisfy Federal agency requirements. The functionality specified will be incorporated into a planned Federal Information Processing Standard (FIPS) Data Dictionary System (DDS). The complete FIPS DDS also will contain additional specifications for such things as the user interface. Comments are being solicited from Federal agencies and suppliers of data dictionary software to determine any modifications that should be made to the Functional Specifications. Information about the effort to develop the planned FIPS DDS and a Management Overview of the Functional Specifications appear in Part I of this document. The Functional Specifications are in Part II.

Key words: Computer program; data dictionary system; data inventory; data management; data standards; database; database management system; documentation; Federal Information Processing Standards Publication; requirements; software.

PART I

1. INTRODUCTION

1.1 Background

As the world's largest user of information processing technology, the Federal government is highly dependent on the use of this technology for carrying out government-wide programs and delivering essential public services. Accordingly, data management software is a tool of rapidly increasing importance which merits special attention in its acquisition and use. The Institute for Computer Sciences and Technology (ICST), in its role as information technology standards-maker for the Federal government, is addressing the need for standards and guidelines in this area through its Data Management Program.

A key software component for the management of information resources is the Data Dictionary System (DDS). A Data Dictionary System is a computer software system that provides facilities for recording, storing, and processing information about an organization's significant data and data processing resources. But, while the DDS alone is a valuable aid to organizing and maintaining data, its utility is severely limited if attention is not paid to such important attributes as integrity, portability, and intercommunications.

ICST is developing a Federal Information Processing Standard (FIPS) Data Dictionary System. The FIPS DDS will be a software specification which Federal agencies may use for procurement purposes in conjunction with Federal Property Management Regulations (FPMR). The FIPS DDS Specifications will not require an agency to use a data dictionary or to use one in a prescribed manner. This report, an interim publication, contains specifications for the basic functions that software supporting a DDS must perform to satisfy Federal requirements.

1.2 Expected Benefits

Together with the necessary management support and controls, a DDS can help Federal agencies:

- o Control and manage their information resources.
- o Increase inter- and intra-agency communication and sharing of information resources.
- o Eliminate redundant data collection.
- o Develop and modify products throughout a system life cycle.
- o Standardize data elements.
- o Document pertinent information about databases, files, computer programs, and manual and automated systems.

A FIPS for a Data Dictionary System will provide additional benefits. In addition to containing standard specifications which can be used in the selection, evaluation, and procurement of DDS software, the FIPS DDS will aid in the portability of the DDS contents. Portability is the ability to transfer data from one DDS to another, without being required to:

- o Recreate or re-enter data descriptions, except by an unload/reload process; or
- o Modify significantly the DDS application that is being transported.

The FIPS DDS also will support portability of acquired skills. Agency personnel will not need additional training to learn new user languages in order to use another DDS.

1.3 ICST Project Approach and Status

1.3.1 Project Approach. The objective of the ICST project is to develop specifications that will support Federal agency requirements, and that will be implemented by a wide spectrum of software suppliers and thus will be available "off-the-shelf." The project, which was initiated late in fiscal year 1979, is based on the following approach:

1. Close and continuing interaction with Federal users to determine which specific capabilities are required by a sufficiently large segment of the Federal community.
2. In-depth technological assessments and intensive consultation with hardware and software vendors, the research community, and Federal developers of in-house data dictionary systems, to determine:
 - o Whether it is technologically practical to develop a particular capability in the near future, i. e. next 3 to 5 years; and
 - o If technically feasible, whether it is economical for the software industry to produce such a capability in a competitive market.
3. Solicitation of comments and suggestions from all affected communities throughout the entire developmental process by issuing periodic reports and conducting workshops.
4. Continuing interchange with the American National Standards Institute (ANSI) Technical Committee, X3H4, to ensure consistency with the planned national standard for a DDS, named the Information Resource Dictionary System (IRDS).
5. Coordination with the Office of Management and Budget (OMB) to:
 - o Evaluate the prototype Federal Information Locator System (FILS) that OMB adopted in implementing the Paperwork Reduction Act of 1980 (P.L. 96-511); and
 - o Ensure that the planned FIPS DDS is compatible with FILS when it becomes fully operational.

1.3.2 Project Status. The work plan for developing a FIPS DDS was divided into the following five phases:

1. State-of-the-art assessment of DDS technology.
2. Requirements Definition.
3. Development of preliminary DDS Functional Specifications.
4. Development of complete FIPS DDS Specifications.
5. Specification of Optional FIPS DDS Module(s) to provide additional capabilities.

During the first phase, relevant literature and existing commercial and Federally-developed data dictionary systems were analyzed. Features and capabilities in the current generation of DDS's were identified. A preliminary assessment identified projected technological trends and issues that warranted further investigation. The following two products were published during the first phase:

1. Prospectus for Data Dictionary System Standard, NBSIR 80-2115. The Prospectus discusses the use of data dictionaries and describes ICST's plans to develop a Federal Information Processing Standard for Data Dictionary Systems. ICST encouraged technical input regarding the appropriate content for a FIPS DDS.
2. Guideline for Planning and Using a Data Dictionary System, Federal Information Processing Standard Publication (FIPS PUB) 76. This publication discusses the capabilities and uses of data dictionary systems. It also provides Federal agencies with basic guidance on DDS selection, planning for the use of a DDS, DDS implementation, and operational usage of a DDS.

In the second phase of the project, interviews were conducted with Federal agencies to identify current and projected requirements for data dictionary software. Interview results, as well as comments received on the Prospectus, are summarized in Federal Requirements for a Federal Information Processing Standard Data Dictionary System (NBSIR 81-2354). This report was disseminated, in the fall of 1981, to Federal agencies. The report also was sent to suppliers of data dictionary software and other individuals and organizations in the private sector who are working with data dictionaries.

Using the results of the first two phases, ICST worked closely with the ANSI X3H4 Technical Committee and with nationally recognized experts on data dictionary systems to develop the DDS Functional Specifications that appear in Part II of this report. In addition, three Federal agency workshops were held to review and refine Federal requirements for the planned FIPS DDS. Invitations to the workshops were extended to Federal agency representatives who were knowledgeable about data dictionary concepts, existing capabilities, and uses of a DDS. Thirty-eight people from the following Federal agencies attended one or more of the workshops:

- Department of Agriculture
- Department of Commerce
 - National Bureau of Standards
 - National Oceanic and Atmospheric Administration
- Department of Defense
 - Office of the Assistant Secretary of Defense (Comptroller)
 - Defense Communications Agency
 - Defense Intelligence Agency
 - Department of the Air Force, Headquarters
 - Department of the Army, Concepts Analysis Agency
 - Military Sealift Command
 - National Security Agency
- Department of Energy
- Department of Health and Human Services
 - Social Security Administration
- Department of Interior
 - Office of the Secretary
 - Bureau of Land Management
 - Bureau of Mines
- Department of Labor
 - General Services Administration
 - Library of Congress
 - National Aeronautics and Space Administration
 - Small Business Administration
 - U.S. Postal Service
 - U.S. Treasury
 - Bureau of Government Financial Operations
 - Internal Revenue Service
- Veterans Administration

Presentations and discussions at the first workshop focused on some of the underlying issues and the preliminary conclusions from the Federal Requirements for a Federal Information Processing Standard Data Dictionary System. At this workshop, the Federal attendees agreed upon such things as the integrity rules and audit requirements that should be included in the Functional Specifications. The second and third workshops focused on the structural model and

functionality of the specifications. The DDS Functional Specifications that appear in this report contain preliminary material reviewed during the workshops with the modifications that Federal representatives felt were needed to satisfy agency requirements. This document represents the results of the third phase of work.

Work on the fourth phase started in October, 1982. Two products, which will constitute the FIPS DDS, are scheduled for development during this phase. These products are the "DDS User Manual" and the "DDS Implementors' Manual." The "DDS User Manual" will contain the specification of the user interfaces. These specifications will include the complete syntax and semantics of all commands, except for features that are specified to be implementor defined. Also included will be specifications for a "help" facility, possible error conditions, the resulting error messages, and the actions to be taken in case of errors.

The "DDS Implementors' Manual" will contain guidance that is to be observed by an implementor. This guidance will be in part an extract of the DDS Functional Specifications. ICST expects that additional modifications will be made to these Functional Specifications as a result of a workshop conducted for suppliers of data dictionary systems, interviews with DDS vendors, additional Federal agency workshops, and comments received on this report. Current plans are to publish the complete FIPS DDS in Fiscal Year 1985. An additional interim report is scheduled to be published in Fiscal Year 1984 to obtain comments on the planned final draft of the FIPS DDS.

1.4 Scope of Report

Readers of the remainder of this report are presumed to be familiar with general data processing concepts and with the concepts and purpose of a data dictionary system. Readers are referred to the earlier publications cited in this Introduction for an overview of the concepts, purpose, and capabilities of DDS's.

A Management Overview of the DDS Functional Specifications appears in Part I, Chapter 2. The DDS Functional Specifications appear in Part II of this report.

2. MANAGEMENT OVERVIEW

2.1 BACKGROUND

Ease of use and flexibility of use are the two major Federal requirements that were identified during both the Federal interviews and the Federal agency workshops. Two prime areas of data dictionary use also were identified. The most important use is and will continue to be oriented toward data management, i.e., to inventory, describe, and standardize data. In the second area, many agencies use a DDS to help manage their ADP resources and provide support for requirements analysis, ADP systems planning and design, change-impact analysis and documentation. An important criterion for an acceptable DDS is a user interface that is easy to learn and use.

Federal agencies also will use the FIPS DDS in a variety of hardware and software environments. To provide the needed flexibility, a "core" DDS and optional modules have become basic concepts. The DDS Functional Specifications represent the "core" DDS. They provide basic support for the prime areas of use identified in the Federal government and can be implemented on small computers as well as medium and large-size computer systems. (The feasibility of implementing the specifications on a small computer system is addressed in Appendix A.) Each additional module will contain more advanced features which will result in a more powerful and complex DDS. An example of a future DDS optional module is an interface to Database Management Systems that conforms to standards being developed by the American National Standards Institute and ICST.

The Functional Specifications are based on an entity-relationship-attribute (E-R-A) structure. For the purposes of this document, the following definitions are used for these terms:

Entity - any named concept, object, person, event, process, or quantity that is the subject of stored or collected data. An entity-type is a class of entities that have the same attributes.

Relationship - a pre-determined ordering between pairs of entities.

Attribute - a property or characteristic of an entity.

A DDS entity represents or names an object, person, etcetera, but it is not the actual data that exists in a file or database. Thus, a DDS entity might be "social-security-number" or "payroll record." It would not be the actual social security number "123-45-6789" or the actual content of a payroll record. Similarly, an attribute represents a characteristic of an entity. An example of an attribute of "social-security-number" is length, e.g., 9 characters. An example of a relationship is "payroll record contains social-security-number."

In order to provide additional flexibility in the use of the "core" data dictionary, capabilities are specified to customize or augment the types of entities, relationships, and attributes that are delivered as part of every software package that conforms to the FIPS DDS. Specification of this customization capability involves several "levels," with a higher level representing a more abstract concept. A rather precise formalism is used in the language of the Functional Specifications to denote the level being addressed. The levels are discussed in Section 2.3 and illustrated in Table 1. Most users of a FIPS DDS will not be aware of the higher levels but Federal agencies will need someone knowledgeable about these higher levels to act as a Dictionary Administrator. Responsibilities of the Dictionary Administrator would include customizing the DDS to satisfy unique agency requirements.

The command language presented in the DDS Functional Specifications is used to illustrate the DDS functionality. The entire syntax is hypothetical, and no indication whatsoever is intended that this syntax will be used in the FIPS DDS. As discussed in the Introduction, questions regarding the user interface are being addressed in the current development effort.

2.2 FUNCTIONAL SPECIFICATIONS FOR THE FIPS DDS

The DDS has three major components:

1. the Dictionary -- the data contained within the DDS.
2. the Dictionary Schema
-- a description of the generic structure of the dictionary.

3. the Dictionary Processing System -- the set of programs which interact with the dictionary and dictionary schema to provide the functionality of the DDS.

2.3 STRUCTURE OF THE DDS

2.3.1 The Dictionary.

The basic unit is the entity. Relationships connect pairs of these entities, and both entities and relationships have attributes assigned to them--values representing properties or characteristics.

A small subset of the dictionary might conceptually have the form given in Figure 1.

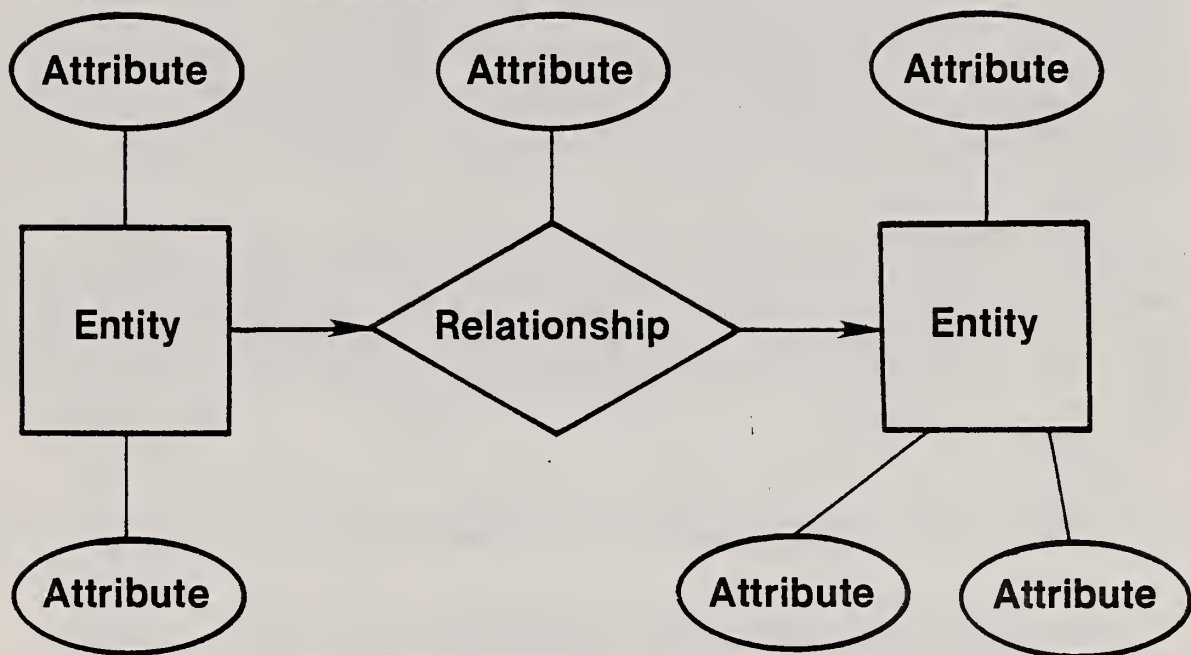


Figure 1

Figure 2 illustrates a specific (and totally hypothetical) example of the typical Entity-Relationship-Attribute structure given in Figure 1.

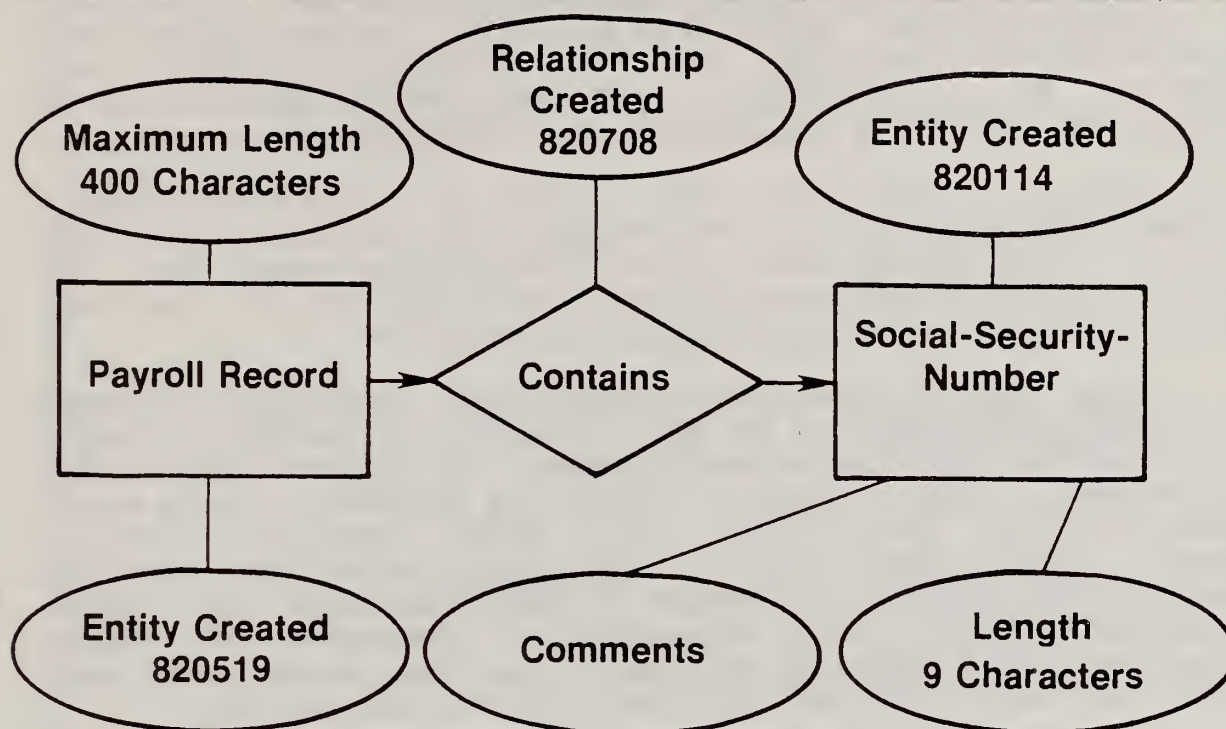


Figure 2

* The dates are in compressed format. This date represents May 19,1982.

2.3.2 The Dictionary Schema.

Attributes can be organized into sets called attribute-types, so that each member of a set represents a like characteristic. For example, "DATE CREATED" is a typical attribute-type.

Similarly, entities can be organized into entity-types. All instances of a specific entity-type have similar or identical characteristics or attribute-types. "Social-security-number" is an example of an "element" entity-type.

In the same manner, relationships can be grouped into relationship-types. All relationships, which are instances of a relationship-type, have attributes from the collection of attribute-types associated with that relationship-type. "System-contains-program" and "record-contains-element" are examples of relationships. (The concept of "type" is generally regarded as a collection of "instances.")

These "types" form the basis of the dictionary schema--the collection of structures that describes the dictionary. The schema also contains structures used for the validation of attributes in the dictionary and for the support of the DDS status and staging facilities.

This entity-relationship-attribute construction used for the dictionary can be used to model the schema as well. Thus the DDS contains a "meta-schema," or schema describing the schema. (The concept of "meta" is defined as data about data.) At this "meta" level, the three concepts "entity-type," "relationship-type," and "attribute-type" are all "meta-entity-types." Instances of these concepts are "meta-entities" which are conceptually connected by "meta-relationships." "Meta-attributes" can be associated with both the "meta-entities" and the "meta-relationships."

Table 1 gives examples of typical entries at each level. The first row of table entries ("Entity-Type," "Element," and "Social-Security-Number") are used to identify social security number. Data about "Social Security Number" appears at the Dictionary Level. Individual social security numbers, e.g., "123-45-6789" and "678-92-1234," are not in the dictionary - they are in a separate file or database. "Element" appears at the Dictionary Schema Level to denote that social security number is a data element. At the highest level, the Schema Model Level, "Entity-Type" indicates that social security number is an entity and not a relationship or attribute. More elaborate examples of the use of meta-attributes are given in Part II, Appendix B.

<u>SCHEMA MODEL LEVEL</u>	<u>SCHEMA LEVEL</u>	<u>DICTIONARY LEVEL</u>
Typical Meta-Entity-Types	Typical Entity-Types, Relationship-Types, and Attribute-Types	Typical Entities, Relationships, and Attributes
Entity-Type	Element <hr/> Record <hr/> Document	Social-Security- Number Agency-Name <hr/> Employee Record Payroll Record <hr/> Form 1040 FIPS Guideline
Relationship-Type	Record-Contains- Element	Payroll-Record- Contains- Employee-Name
Attribute-Type	Length <hr/> Creator	9 Characters <hr/> ADP Division

Table 1

2.3.3 Customizing the DDS.

All the formalism at the meta-level is necessary for specifying the most important use of the meta-structures--the customization of the data dictionary system.

Certain entity-, relationship-, and attribute-types are standard within the DDS (see Section 2.4.1.), but agencies may need to augment this collection in order to satisfy their unique requirements. The DDS facility that supports customization uses the meta-structures. The Dictionary Administrator has a full set of commands available to invoke the customization facility in order to modify or augment the "system standard schema."

2.4 THE SYSTEM STANDARD SCHEMA

In order for the FIPS DDS to provide Federal agencies with the full benefits of inter- and intra-agency communication, the specifications include a specific collection of entity-types, relationship-types, and attribute-types. This collection, called the "system standard schema," is expected to be delivered as part of every software package conforming to the Federal Information Processing Standard. An agency can then augment this collection using the customization facility discussed in Section 2.3.3.

2.4.1 System Standard Entity-Types. Reflecting DDS usage in the Federal government, the system standard schema provides for eight data, process, and external entity-types:

Data Entity-Types

1. ELEMENT, to describe instances of data belonging to an organization. Typical ELEMENTs are "social security number" and "agency name."
2. DOCUMENT, to describe instances of human readable data collections. Typical DOCUMENTs are "Form 1040" and "FIPS Guideline."
3. RECORD, to describe instances of logically associated data. Typical RECORDs are "employee record" and "payroll record."
4. FILE, to describe instances of an organization's data collections. Typical FILEs are "roster" and "accounts receivable."

Process Entity-Types

5. SYSTEM, to describe instances of collections of processes and data. Typical SYSTEMs are "personnel system" and "airline reservation system."
6. PROGRAM, to describe instances of automated processes. Typical PROGRAMs are "roster update" and "COBOL compiler."
7. MODULE, to describe instances of automated processes which are either logical subdivisions of program entities or independent processes which are called by program entities. Typical MODULEs are "sort records" and "main program."

External Entity-Types

8. USER, to describe members belonging to an organization who use or are responsible for data in the data dictionary system. Typical USERS are "John Doe" and "personnel division."

Additionally, DICTIONARY-USER (to identify individuals and access privileges) and ACCESS-CONTROLLER entity-types are specified for the Dictionary Administrator to use in the management of the DDS's security system.

2.4.2 System Standard Relationship-Types.

The collection of relationship-types provided by the system standard schema is summarized by Table 4-1 on pages 4-12 through 4-15 of the DDS Functional Specifications. This collection of 50 relationship-types includes virtually all the connections between system standard entity-types that might prove useful to most agencies most of the time.

Most of these relationship-types are themselves grouped into classes, although there are a few such as STANDARD-FOR that are not included in a class. The six main classes are:

1. CONTAINS, to describe instances of an entity being composed of other entities. A typical CONTAINS relationship-type is RECORD-CONTAINS-ELEMENT, which has as a possible instance the relationship "Payroll-record-contains-employee-name."
2. PROCESSES, to describe associations between DATA and PROCESS entity-types. A typical PROCESSES relationship-type is SYSTEM-PROCESSES-FILE, which has as a possible instance the relationship "budget-system-processes-cost-center-file."

3. RESPONSIBLE-FOR, to describe associations between entities representing organizational components and other entities to denote organizational responsibility. A typical RESPONSIBLE-FOR relationship-type is USER-RESPONSIBLE-FOR-DOCUMENT, which has as a possible instance the relationship "personnel-office-responsible-for-SF-171."
4. RUNS, to describe associations between USER and PROCESS entity-types, illustrating that a person or organizational component is responsible for running a certain process. A typical RUNS relationship-type is USER-RUNS-PROGRAM, which has as a possible instance the relationship "John-Doe-runs-system-backup."
5. TO, which describes "flow" associations between PROCESS entity-types. A typical TO relationship-type is MODULE-TO-MODULE, which has as a possible instance the relationship "main-program-to-sort-routine," (indicating flow of control or data within a program).
6. DERIVED-FROM, describing associations between entities where the target entity is the result of a calculation involving the source entity. A typical DERIVED-FROM relationship-type is DOCUMENT-DERIVED-FROM-FILE, which has as a possible instance "annual-report-derived-from-plans-file."

2.4.3 System Standard Attribute-Types.

The attribute-types developed for inclusion in the system standard schema are the ones that agencies generally want applied to system standard entity-types. Among the 35 attribute-types in this collection are some that are common to all entity-types, including

- o attribute-types that provide audit trail information. A typical audit attribute-type is DATE-CREATED, which has as a possible instance "810101."
- o attribute-types that provide general documentation for entities, for example DESCRIPTION, and CLASSIFICATION.

Other system standard attribute-types are associated with just one or a few entity-types. For example, ACCESS-METHOD, with possible attribute instance "indexed sequential," is unique to the FILE entity-type.

Table 4-2 on pages 4-66 and 4-67 of the DDS Functional Specifications summarizes these associations.

As an additional feature of the system standard schema, certain relationship-types have attribute-types associated with them. For example, the attribute-type REL-POSITION, associated with the RECORD-CONTAINS-ELEMENT relationship-type, can be used to document the relative position of an ELEMENT within a RECORD. Thus, "3" might be the REL-POSITION attribute that applies to the "employee-record-contains-social-security-number" relationship.

2.5 THE DDS COMMAND LANGUAGE

The DDS command language, designed merely to illustrate the DDS's functionality, contains facilities for interacting with both the dictionary schema and the dictionary itself.

2.5.1 Interaction with the Dictionary Schema.

The commands available for interaction with the schema fall into two broad categories:

Schema Maintenance Commands

To add a new entity-, relationship-, or attribute-type, a dictionary administrator would use the CREATE-META-ENTITY command, since the above types are all meta-entities. To modify or delete a type the ALTER-META-ENTITY and ABOLISH-META-ENTITY commands are used, respectively. For example, ABOLISH-META-ENTITY DATABASE would delete an already existing DATABASE entity-type and

```
CREATE-META-ENTITY  
ATTRIBUTE-TYPE WEIGHT
```

would create a new attribute-type called WEIGHT.

The mechanism that specifies that a particular attribute-type is associated with a particular entity- (or relationship-) type is the meta-relationship. Thus, once a new attribute-type has been created it is assigned to the appropriate entity-type by using the CREATE-META-RELATIONSHIP command.

```
CREATE-META-RELATIONSHIP  
EQUIPMENT AND WEIGHT
```

assigns the attribute-type WEIGHT to the entity-type EQUIPMENT. The ALTER- and ABOLISH-META-ENTITY commands are

analogous.

Schema Reporting Commands

Since these commands are designed for the use of the Dictionary Administrator, the reports generated are normally general listings and catalogs of the schema entries. The commands include META-CATALOG, which produces a report on the entire contents of the schema; META-LIST, which lists all valid meta-attributes and meta-relationships of the specified meta-entities; and META-TRACE, which produces a report on all meta-entities which are related (via a meta-relationship) to a specified meta-entity.

2.5.2 Interaction with the Dictionary.

The facilities available for modifying and reporting on the dictionary content are more elaborate than those for interacting with the schema. In particular, the Dictionary Administrator or dictionary user can use fairly powerful search criteria to identify dictionary entities for subsequent reporting or manipulation. This process, called qualification, results in a list of the names of the desired entities. The resulting qualification list can be used as input to other commands.

Qualification

The eight categories of qualification (combinations of which can be used in a command) are:

1. Selection by entity-type-name (e.g. FILE) or by specific entity-name (e.g. "payments-receivable");
2. Selection by using an alternate name instead of a primary name;
3. Restriction of entity selection based on some characteristic of the primary name, such as length (e.g. "LENGTH = 10") or a specific character string in the primary name (e.g. "STARTS-AS prog");
4. Restriction based on whether or not a specific entity is connected to another through a named relationship-type (e.g. "payroll-record CONTAINS social-security-number");
5. Restriction based on the name of the person who created or last changed the entity, the date the entity was created or last changed, or on the number of changes to the entity that have been made;

6. Restriction based on the values of specific attributes;
7. Restriction based on the presence of specific character strings in attributes composed of text (e.g. "CONTAINS social-security"). Two such system standard attribute-types are DESCRIPTION and COMMENTS; and
8. Selection through the use of classification keywords (which are the attributes of the attribute-type CLASSIFICATION). The presumption here is that the agency using the DDS first defines a list of "legal" keywords (e.g. a Thesaurus). The dictionary administrator or user then selects CLASSIFICATION attributes from this list when defining entities, and thus the keywords are subsequently available for qualification purposes.

Existing qualification lists, considered as sets of entities, can be acted upon by the union, intersection, and set difference operators to produce new lists. Other facilities for manipulating qualification lists exist as well. These qualification lists also can be named, saved, and later re-used.

Dictionary Maintenance Commands

The dictionary and schema maintenance facilities are quite analogous. For clarity in the DDS Functional Specifications, parallel commands have deliberately been given very different names in the hypothetical command language. Thus one would CREATE-META-ENTITY at the Dictionary Schema Level and ADD-ENTITY at the Dictionary Level. Likewise, one would ALTER-META-RELATIONSHIP at the Dictionary Schema Level and MODIFY-RELATIONSHIP at the Dictionary Level. This does not imply that separate, parallel commands will be used in the final user interface.

Dictionary Reporting Facilities

These commands perform the central function of producing reports on the contents of the dictionary. Three reporting commands of special interest are:

1. The CATALOG command to report on selected entities and specified relationships of these entities by showing the attributes of each. The full qualification facility is available. A typical CATALOG command might be

CATALOG
FILE-A, FILE-B
SEQUENCE: ENTITY-NAME

which produces a report giving the names FILE-A and FILE-B, together with all the attributes of these two entities, and

CATALOG
QUALIFY ENTITY-TYPE
PROGRAM, MODULE
LANGUAGE = COBOL
SEQUENCE: ENTITY-NAME

which produces a list of all programs and modules written in COBOL.

2. The LIST command, which produces a report containing the primary names of entities of designated entity-types.
3. The IMPACT-OF-CHANGE command, which identifies all entities in the dictionary which might be affected in some manner by changes to designated entities (chosen by qualification). For example,

IMPACT-OF-CHANGE
QUALIFY ENTITY-TYPE ELEMENT
FILE-CONTAINS-ELEMENT
REPORT-DERIVED-FROM-FILE
SYSTEM-PROCESSES-FILE
SEQUENCE: REPORT-SET ENTITY-NAME

This command generates a list of:

1. All element names;
2. For each of these elements, all file names that contain occurrences of this element (i.e. connected by a FILE-CONTAINS-ELEMENT relationship); and
3. For each of these files, the names of all reports that are derived from and all systems that process the file (i.e. connected to the file by REPORT-DERIVED-FROM-FILE and SYSTEM-PROCESSES-FILE relationships, respectively)

In short, this report would tell the user precisely which files, reports, and systems (as documented in the DDS) would be affected by a change to each element.

Dictionary Query Commands

A query facility is included as an important part of the DDS Functional Specifications. While the dictionary reporting capabilities can be used to generate a short report which may be returned on-line to a user terminal, the use of the report facility is expected to be less flexible than the query facility. The simple queries, used to generate simple outputs, use the qualification facilities as basic building blocks, but also use special keywords to simplify the user's task. For example, both

```
QUERY
QUALIFY
ALL
STRING = "person"
```

and

```
QUERY
STRING "person"
IN PRIMARY-NAME
```

return the names of all entities that contain the string "person" in their primary-name.

To prevent an unexpectedly large output from being returned to the user terminal every query command first returns a count of qualified entities to the user. The user can then direct further query output to an alternate location.

2.6 THE DDS SOFTWARE INTERFACES

The DDS Functional Specifications include three interfaces with other software systems:

1. There are commands to produce, from the dictionary, representations of data usable within an ANSI COBOL program. In particular, this facility can produce a DATA DIVISION, including File and Working Storage Sections.
2. There are commands that transfer a selected portion of the contents of one dictionary which conforms to the FIPS DDS to another FIPS dictionary. This "export/import" capability improves the portability of dictionaries, but it does involve a nontrivial transfer--two different schemas are involved. Such a transfer might cause serious integrity problems,

so the specifications require that the basic structures of the schemas corresponding to the two dictionaries be "essentially" identical.

3. There are commands that provide access to a dictionary from a program written in any standard language that has a CALL facility.

2.7 SECURITY FACILITIES OF THE FIPS DDS

The DDS security feature consists of three levels of access control:

1. The first level controls the access to both the Dictionary Processing System and to specified dictionaries. This level is provided by the implementor of the DDS software in a manner which is an option of the implementor.
2. The second level of control, the "global" level, occurs through dictionary entities of the type DICTIONARY-USER and their attributes. These entities specify the permissions that have been granted to a user in terms of the commands that the user can execute against specific entity-types in a controlled or uncontrolled status, in the dictionary. Likewise, privileges can be assigned to specified relationship-types. Attributes also are used to specify privileges of a dictionary user with respect to the schema.
3. The third level of control, the "local" level, occurs through dictionary entities of the type ACCESS-CONTROLLER and their attributes. Any entity can be protected at this level by establishing a relationship in the dictionary between that specific entity and an entity of the type ACCESS-CONTROLLER. As part of the creation of an ACCESS-CONTROLLER entity, the DDS also creates an attribute of type READ-LOCK and an attribute of type WRITE-LOCK. Corresponding attributes of type READ-KEY and WRITE-KEY can then be assigned to designated users to provide them with keys. The keys allow the execution of specific commands on an entity that has local protection. Global protection has precedence over local protection in the sense that local protection is not checked unless the user has global permission for the command on the entity-type in question.

PART II
DDS FUNCTIONAL SPECIFICATIONS

E R R A T A

The July 1982 version of the DDS Specifications, as revised, contains a number of recognized omissions and inconsistencies which have not yet been corrected. This Errata will discuss the major ones.

1. Names of Entities in the Dictionary

The structure of the name of a dictionary entity is discussed in Section 3.4; however, the rules specified there are not adhered to closely in the body of the specification. To recapitulate briefly, an entity has a **primary name** which is unique in the dictionary. This primary name is a concatenation of the **assigned name** and the **version number**.

- o The assigned name of an entity is declared in either an ADD-ENTITY, RENAME, or COPY command. For those commands, the rule stating that "entity-name cannot be the primary name of an entity in the dictionary" should be replaced by "the assigned name specified cannot be the assigned name of an entity in the dictionary". Similarly, for entity-types where it is stated that "entities have system-generated primary names", this should state that "entities have system-generated assigned names".
- o In line with the above, the meta-attribute-types MINIMUM-NAME-LENGTH, MAXIMUM-NAME-LENGTH, PICTURE, and ID-START of meta-entities of type entity-type should refer to the assigned name of an entity instead of, as stated in the specification, the primary name of an entity.
- o In other commands, in general, whenever the primary name of an entity is to be specified, this should be replaced by "the primary name or the assigned name of an entity". If the latter is used, the version number corresponding

to the VERSION-DEFAULT in effect will be used to construct the primary name of the entity being addressed in the command. There are isolated exceptions to this statement.

- o In a number of places in the specification the phrase "multiple versions of an entity" is used. Such statements should be replaced by "a set of entities which have the same assigned name", as the phrase used is meaningless.

2. Attributes of Entities of Type DICTIONARY-USER

The attributes of entities of type DICTIONARY-USER are used to designate the permissions and exclusions for the use of the dictionary system that are in effect for a given user. Such attributes may contain schema descriptors of type entity-type, relationship-type, or status. The specification should state that:

- o It is **not** specified that these attributes, when entered into the dictionary, will be checked as being valid schema descriptors.
- o A schema descriptor may be deleted from the schema even though it exists as an attribute of an entity of type DICTIONARY-USER. Moreover, such a deletion will not affect its existence as an attribute, i.e., the scope of the deletion is limited to the schema. A similar situation is in effect if a schema descriptor is renamed through the use of the command CHANGE-META-NAME.
- o The scope of the Schema Reporting Commands of Section 5.2 is limited to the schema, and as such, does not extend to the attributes of entities of type DICTIONARY-USER. It is intended that dictionary commands be used to report on the attributes of these entities.

3. Meta-entities of Type Attribute-Type-Validation-Data

The current specification states that there may exist at most one meta-entity of type Attribute-Type-Validation-Data for any meta-entity of type attribute-type. This restriction should be replaced by the following:

- o There may exist more than one meta-entity of type Attribute-Type-Validation-Data for a meta-entity of type attribute-type subject to the following condition:

If more than one meta-entity of type Attribute-Type-Validation-Data exists for a meta-attribute of type attribute-type, then the meta-attribute of type VALUE/RANGE must be the same for all of them.

- o If more than one such meta-entity of type Attribute-Type-Validation-Data exists for a meta-attribute of type attribute-type, validity of attributes of this attribute-type will be determined by a logical OR operation, i.e., an attribute will be valid if it is valid according to any one Attribute-Type-Validation-Data meta-entity.

4. The MODIFY-ENTITY Command

The MODIFY-ENTITY Command specified in 6.3.8 can exist in two forms, only the first one of which is given. The second form provides for a set of entities to be modified to be stated in a list-name in the same manner as this is done for the DELETE-ENTITY command. In this form of the command:

- o The version number option is not available.
- o The modification specified in clause-1, ... apply to all the entities included in list-name.

5. Inverse Names of Relationship-types and Relationship-class-types

The DDS Functional Specifications provide the capability for assigning an inverse name to every relationship-type and relationship-class-type, but the commands for interaction with the dictionary do not mention the use of an inverse name.

The rules for each command in which a relationship-type or relationship-class-type is used should be augmented in the following manner:

In place of using the name of a relationship-type or relationship-class-type with members entity-1 (as the first member of the relationship or relationship-class) and entity-2 (as the second member), the inverse name can be used. In this case entity-2 should be specified as the first member and entity-1 as the second member. For example the command

```
ADD-RELATIONSHIP
SYSTEM-CONTAINS-PROGRAM
A-SYSTEM, XY123
```

can equally be expressed as

```
ADD-RELATIONSHIP
PROGRAM-CONTAINED-IN-SYSTEM
XY123, A-SYSTEM
```

CHAPTER 1. INTRODUCTION

Part II of the **DDS FUNCTIONAL SPECIFICATIONS** contains the functional specifications of the core standard Data Dictionary System.

For purposes of the present document, a **Data Dictionary System (DDS)** is defined as a computer software system that provides facilities for recording, storing, and processing information about an organization's significant data and data processing resources.

The DDS has three components:

1. **The Dictionary Schema** which describes the structure of the Dictionary. Whenever no ambiguity exists, the Dictionary Schema will be referred to as the "Schema".
2. **The Dictionary** which is the structured collection of entities and relationships and their associated attributes.
3. **The Dictionary Processing System** which is the set of programs in the DDS which interact with the Dictionary and Dictionary Schema in order to provide the functionality of the DDS.

The specification of the DDS is given in the following seven chapters:

- o Chapter 2 discusses the environment in which the DDS operates, as well as the defaults that exist for the system.

- o Chapter 3 presents the specification of the Dictionary and the Dictionary Schema in terms of their structural characteristics.
- o Chapter 4 contains the specification of the Dictionary Schema that is part of the core standard DDS (i.e. the system-standard schema, as defined below).
- o Chapter 5 contains the specification of the commands that are available for interaction with the Dictionary Schema, including those commands that are available for customization of the Dictionary Schema to an installation and making extensions to it.
- o Chapter 6 contains the specification of the commands that are available for interaction with the Dictionary.
- o Chapter 7 presents the specification of the interfaces of the DDS to other software systems.
- o Chapter 8 contains the specification of the commands and utilities that are required by the Dictionary Administrator. This chapter also contains the specification of the security facilities of the DDS and the description of the commands associated with these facilities.

Consideration must be given to the fact that the syntax used in this document is intended to be only illustrative. No implication whatsoever is intended that this syntax will or should be used in the core standard. It then follows that the entire question of the user interface to the DDS is not addressed in this document. It is recognized that it is intended that the DDS be usable by a wide range of users, some of which will not have a technical data processing background or training. Under these circumstances it is not only desirable, but mandatory, that at least a part of the DDS functionality be available through a

user-friendly interface. This problem is to be addressed at the time the syntax of the DDS is designed. The current document contains only the specification of the functionality of the DDS, and does **not** address the manner in which this functionality can be invoked by a user of the DDS.

The following **Definitions** will be used throughout this specification:

The term **Implementor of the DDS** is used to denote the person or organization producing the software which embodies the functionality of the DDS standard.

The term **Installer of the DDS** is used to denote the person or group of persons working for an organization which has acquired a DDS responsible for placing the DDS on the organization's computer system and making its functionality accessible to the organization. The time at which this activity occurs is referred to as the **Installation Time** of the DDS.

The term **Dictionary Administrator** is used to denote the person or group of persons with the on-going responsibility for making the dictionary system and the data contained in it available to the organization which uses it. Among other duties, the dictionary administrator will be responsible for customization of the dictionary schema at installation time and throughout the life of the dictionary system. It is assumed that the dictionary administrator will also be responsible for the security of the dictionary system and its overall integrity.

The **System-Standard Dictionary Schema** is the dictionary schema that is produced by the implementor of the DDS. The system-standard schema is composed of system-standard entity-types, system-standard relationship-types, system-standard attribute-types, and system-standard control descriptors.

The DDS system contains facilities through the use of which the system-standard schema can be modified. These facilities will be referred to as **Extensibility Facilities**.

A number of examples are presented throughout this document. It is to be emphasized that these examples are not to be considered to be part of the specification and that the only reason for their presence is for illustrative purposes. A major example on the use of extensibility facilities is found in Appendix B.

CHAPTER 2. THE DDS ENVIRONMENT AND DDS DEFAULTS

2.1 THE DDS ENVIRONMENT

[This section will discuss the operating environment of the DDS relative to the computer system and its system software on which the DDS resides.

In particular, a discussion will be given of the functional requirements that the host system software will have to satisfy for the log-on to the dictionary system. As part of the log-on the verification of the user will have to occur, and the results passed on to the DDS.

As part of the log-on, the user will identify the dictionary that is to be worked on by means of the statement

DICTIONARY = dictionary-name

and verification will occur that the user is entitled to access the dictionary named. After completion of work on the named dictionary, a user may, by means of the command

DICTIONARY = dictionary-name-1

access another dictionary under control of the same DDS, where again verification will occur to insure that the user has access to the dictionary named.]

2.2 DDS DEFAULTS

In this section the existing DDS defaults will be discussed. The meaning of these defaults is that they represent conditions which are in effect at the beginning of each session or run unit, and remain in effect until changed. Such a change is effected by the user by issuing a command of the form

DEFAULT-NAME = default-value

Such a command can be issued at any time during a session or run unit immediately preceding a command.

2.2.1 MODE FOR MAINTENANCE OF THE DICTIONARY SCHEMA

All commands that cause a modification to be made to the dictionary schema are executable in one of two modes:

- o CHECK: In this mode the command will only be checked for errors and no update to the schema will take place.
- o UPDATE: In this mode the command, if error-free, will cause a modification to the schema to take place.

The default mode is **CHECK**. The mode can be changed by means of the command

SCHEMA-MODE = UPDATE

and this mode remains in effect for the session or run unit until modified by the command

SCHEMA-MODE = CHECK

2.2.2 MODE FOR MAINTENANCE OF THE DICTIONARY

Similarly to the dictionary schema, all commands that cause a modification to be made to the dictionary are executable in one of two modes:

- o **CHECK:** In this mode the command will only be checked for errors and no update to the dictionary will take place.
- o **UPDATE:** In this mode the command, if error-free, will cause a modification to the dictionary to take place.

The default mode is **UPDATE**. The mode can be changed by means of the command

DICTIONARY-MODE = CHECK

and this mode remains in effect for the session or run unit until modified by the command

DICTIONARY-MODE = UPDATE

2.2.3 DEFAULT STATUS

As discussed in Section 3.1.5, the DDS has a status facility that provides two kinds of statuses for entities in the dictionary. One such status is known as the **CONTROLLED** status, and there exist rules for structural integrity for entities in this status. For example, it is not possible that if the **PAYROLL-RECORD** contains the element **EMPLOYEE-ID**, that **PAYROLL-RECORD** be in the **CONTROLLED** status, and **EMPLOYEE-ID** not be in the **CONTROLLED** status. Whenever a new entity is added to the dictionary, it must be entered as being in a status other than the **CONTROLLED** status. There exists **for each user of the dictionary** a status which exists as the default at the beginning of a session or run unit, such that if the user adds an entity into the dictionary it will be placed into that status. Based on decisions made by the

Dictionary Administrator, there may be a single default status for all users of the dictionary, or a set of these with users being assigned to one of the set based on the tasks that are currently being carried out by the user.

The status with which the user is working may be changed by means of the command

DEFAULT-STATUS = STATUS-NAME

where STATUS-NAME cannot be the name of the CONTROLLED status. Execution of this command is also subject to the security provisions of the DDS. The command will only be allowed to execute if the user of the dictionary has been given access to the status being specified. Details of the security provision of the DDS are given in Chapter 8.

2.2.4 VERSION DEFAULT

Facilities are provided in the DDS for an entity to exist multiple times in the dictionary, these various instances being distinguished from each other by different version numbers which are assigned by the system (and optionally by a user of the DDS) in a chronological manner. The purpose of the version default is to indicate to the DDS which versions of entities the user wishes to work with. The default in effect at the beginning of a session or run unit is for the user to work with the latest version (i.e. the one with the highest version number) of each entity. This can be changed by the command

DEFAULT-VERSION = STATUS-NAME

where STATUS-NAME is the name of the CONTROLLED status, as discussed above. Execution of this command, which is subject to the security provisions discussed in Chapter 8, will indicate that the user wishes to work with entities in the CONTROLLED status. The user may return to the original default with the command

DEFAULT-VERSION = LATEST

Provisions exist in the commands specified in Chapter 6 for the user to select a specific version of an entity by specifying the version number of the entity.

2.2.5 CODE VALUES DEFAULT

As described in Section 3.1.3, facilities exist in the DDS to store in the schema sets of codes and their transliterated values. The option exists for the user to specify whether a report or the output to a query shall contain code values or transliterated values.

The default in existence at the beginning of a session or run unit is for transliterated values to be displayed in a report or response to a query. This default can be changed by the command

CODE-VALUES-DEFAULT = CODE

and can be returned to the original default by the command

CODE-VALUES-DEFAULT = LITERALS

CHAPTER 3. THE DICTIONARY AND DICTIONARY SCHEMA AND THEIR STRUCTURE

In this chapter we will describe the Dictionary and the Dictionary Schema and their structural characteristics. For this purpose we will use an Entity - Relationship - Attribute (E-R-A) model, as follows:

- o There exists a set of entities.
- o There exists a set of relationships, each one of which is an ordered pair of entities.
- o There exists a set of attributes, each one of which is associated with an entity and/or a relationship.
- o Each entity has a unique name, which is referred to as the primary name of the entity.
- o There exists a set of entity-types, such that each entity is of exactly one entity-type.
- o There exists a set of relationship-types, such that each relationship is of exactly one relationship-type.
- o There exists a set of attribute-types, such that each attribute is of exactly one attribute-type.
- o Each entity-type, relationship-type, and attribute-type has a unique name.

We will use the following terminology throughout this document:

- o An entity, relationship, or attribute will be referred to as a **Descriptor**.
- o An entity-type, relationship-type, or attribute-type will be referred to as a **Schema Descriptor**.

There exists a potential ambiguity and resultant confusion in the application of these terms to both the Dictionary and Dictionary Schema. In order to clarify which one of these levels is being discussed, the "entities" in the Dictionary Schema will be referred to as "meta-entities", the "relationships" between these are referred to as "meta-relationships", and the "attributes" existing for both of these as "meta-attributes". Similarly, the types of these will be referred to as "meta-entity-types", "meta-relationship-types", and "meta-attribute-types".

3.1 THE STRUCTURE OF THE DICTIONARY SCHEMA

The structure of the Dictionary Schema is described in terms of:

- a) a set of meta-entity-types
- b) a set of meta-relationship-types
- c) a set of meta-attribute-types

such that:

1. A meta-relationship-type is a named ordered pair, each member of which is a meta-entity-type.
2. Every meta-attribute-type is associated with a meta-entity-type and/or a meta-relationship-type.
3. Every meta-entity-type, meta-relationship-type, and meta-attribute-type has a unique name.

Since the Dictionary Schema describes the structure of the Dictionary it then follows that:

Every Schema Descriptor is a meta-entity.

The order of presentation in this section is as follows:

- o We will first specify the meta-entity-types that exist in the structure of the dictionary schema.
- o This will be followed by the specification of the meta-relationship-types that exist in the structure of the dictionary schema.
- o We then specify the meta-attribute-types that exist for meta-entity-types, followed by the specification of meta-attribute-types for meta-relationship-types.
- o A limited number of examples are given within this specification; the bulk of the examples dealing with this subject is deferred to Appendix B.

3.1.1 META-ENTITY-TYPES

The following meta-entity-types exist in the structure of the dictionary schema:

1. ENTITY-TYPE

Meta-entities of this type correspond to entity-types, instances of which are in the dictionary.

2. RELATIONSHIP-TYPE

Meta-entities of this type correspond to relationship-types, instances of which are in the dictionary.

3. RELATIONSHIP-CLASS-TYPE

Meta-entities of this type correspond to relationship-class-types (i.e., sets of relationship-types), instances of which are in the dictionary.

4. ATTRIBUTE-TYPE

Meta-entities of this type correspond to attribute-types, instances of which are in the dictionary.

5. ATTRIBUTE-GROUP-TYPE

Meta-entities of this type correspond to attribute-group-types (i.e., ordered tuples of attribute-types, instances of which are in the dictionary).

6. ATTRIBUTE-TYPE-VALIDATION-PROCEDURE

Meta-entities of this type represent procedures that exist for validation of attributes of a given attribute-type.

7. ATTRIBUTE-TYPE-VALIDATION-DATA

Meta-entities of this type represent data that is used in conjunction with attribute-type-validation-procedures.

8. STATUS-NAME

A status represents the condition or quality of an entity. Meta-entities of this type name the statuses that exist in the dictionary, and represent the rules that exist for status integrity of structures.

9. STAGE-NAME

A stage is a subdivision of the System Life Cycle. Meta-entities of this type name the stages that exist in the dictionary.

The meta-attribute-types that pertain to each one of these meta-entity-types will be discussed in 3.1.3.

Examples of the foregoing are the following:

SYSTEM, FILE, and ELEMENT are instances of the meta-

entity-type ENTITY-TYPE.

FILE-CONTAINS-RECORD and RECORD-CONTAINS-ELEMENT are instances of the meta-entity-type RELATIONSHIP-TYPE.

LANGUAGE (as it pertains to a program) and ALTERNATE-NAME of an element are instances of the meta-entity-type ATTRIBUTE-TYPE.

A relationship-class-type is a set of relationship-types. The utility of this construct is that it permits sets of relationship-types to be grouped in terms of a single concept, with the individual relationship-types only differing in terms of the entity-types involved. As an example, the concept of "CONTAINS" is applicable to a variety of entity-types, such as a FILE "CONTAINS" a RECORD, or a RECORD "CONTAINS" an ELEMENT. Here "CONTAINS" is a relationship-class-type composed of the given relationship-types. Relationship-class-types also have significance from the point of view of the structure of the command language. For purposes of simplicity and ease-of-use it is desirable to allow users of the DDS to use names of relationship-class-types, as opposed to forcing them to use the names of the relationship-types, of which there are many more than names of relationship-class-types.

The concept of an attribute-group-type is introduced to accomodate the requirement that in some cases two or more attributes need to be viewed as an ordered set. The simplest such case is where attributes are needed to describe a set of lower and upper bounds, and it is then necessary to be able to specify that the first lower bound and the first upper bound go together, the second lower bound and the second upper bound go together, etc. In this case an attribute-group-type called RANGE can be introduced whose (ordered) members are attribute-types LOWER BOUND and UPPER BOUND. It should be noted that if so desired, a user may view each attribute-type in the attribute-group-type by itself, i.e. as an attribute-type of the schema element. Thus, in the example used, the attribute-type UPPER-BOUND, for instance, may be queried outside the context of the attribute-group-type RANGE. Update operations on such attribute-types will, in gen-

eral, however, be restricted to the context of the attribute-group-type, unless one of the attribute-types has a meaning by itself outside the context of the attribute-group-type. This is not the case in the example we have been dealing with, but does occur in the attribute-group-type IDENTIFICATION-NAMES which we will discuss later.

Both ATTRIBUTE-TYPE-VALIDATION-PROCEDURE and ATTRIBUTE-TYPE-VALIDATION-DATA meta-entities serve the purpose of allowing that attributes of a given entity-type can be restricted to either a predefined list of values or ranges.

The meta-entity-type STATUS-NAME permits the quality of dictionary entities to be recorded; for example the STATUS-NAME called OPERATIONAL might be used to designate that a SYSTEM and its constituents are being used in a "production" environment.

An instance of the meta-entity-type STAGE-NAME, for example DESIGN, allows entities in the dictionary to be characterized as being in the design stage, which in this case is a recognized phase of the system life cycle.

3.1.2 META-RELATIONSHIP-TYPES

A meta-relationship-type is a named ordered pair of meta-entities. We use the notation

$$M-R-T(\text{meta-entity-name-1}, \text{meta-entity-name-2})$$

to denote a meta-relationship-type which has as members the meta-entities with names meta-entity-name-1 and meta-entity-name-2. No ambiguity will be found to exist in this notation, as there will exist no more than one meta-relationship-type with a given pair of meta-entities as members.

Meta-attribute-types that apply to these meta-relationship-types will be given later in 3.1.4.

The following meta-relationship-types exist in the structure of the dictionary schema (a brief statement of their usage being given for illustrative purposes):

1. M-R-T(RELATIONSHIP-TYPE, ENTITY-TYPE)

Serves to specify the entity-types that are members of a relationship-type.

2. M-R-T(ENTITY-TYPE, ATTRIBUTE-TYPE)

Serves to specify that an attribute-type pertains to an entity-type.

3. M-R-T(RELATIONSHIP-TYPE, ATTRIBUTE-TYPE)

Serves to specify that an attribute-type pertains to a relationship-type.

4. M-R-T(RELATIONSHIP-CLASS-TYPE, RELATIONSHIP-TYPE)

Serves to specify that a relationship-type is included in a relationship-class-type.

5. M-R-T(ATTRIBUTE-GROUP-TYPE, ATTRIBUTE-TYPE)

Serves to specify that an attribute-type is a member of an attribute-group-type.

6. M-R-T(ENTITY-TYPE, ATTRIBUTE-GROUP-TYPE)

Serves to specify that an attribute-group-type pertains to an entity-type.

7. M-R-T(RELATIONSHIP-TYPE, ATTRIBUTE-GROUP-TYPE)

Serves to specify that an attribute-group-type pertains to a relationship-type.

8. M-R-T(ATTRIBUTE-TYPE, ATTRIBUTE-TYPE-VALIDATION-PROCEDURE)

Serves to indicate that an attribute-type-valida-

tion-procedure is to be used for an attribute-type.

9. M-R-T(ATTRIBUTE-TYPE, ATTRIBUTE-TYPE-VALIDATION-DATA)

Serves to indicate the list of values or ranges that are used in the validation of the attributes of an attribute-type.

3.1.3 META-ATTRIBUTE-TYPES OF META-ENTITY-TYPES

The following meta-attribute-types apply to all meta-entities of type **ENTITY-TYPE**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the

schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

SYSTEM-LOCK

An indicator, which, when it has the value ON, will disallow any attempt to delete this meta-entity.

INSTALLATION-LOCK

Similar to SYSTEM-LOCK, but special commands are available to override this lock.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

MINIMUM-NAME-LENGTH

The minimum number of characters which the primary name of an entity of the given entity-type must have.

MAXIMUM-NAME-LENGTH

As above, giving the maximum length.

PICTURE

The picture of the primary name of an entity. Multiple meta-attributes are allowed, in which case the name must be one of the representations given.

SYSTEM-GENERATED

The primary name of an entity is generated by the system. In this case a value of the meta-attribute-type ID-START must be given; the first name generated

will be this value and subsequent names will be formed by incrementing the preceding name by 1.

ID-START

The starting value for system-generated primary names. It must be of the form A...N..., where A denoted any alphanumeric character and N is an integer. If this option is used, any values specified for the meta-attribute-types MINIMUM-NAME-LENGTH, MAXIMUM-NAME-LENGTH, and PICTURE are ignored.

CONNECTABLE

Specifies whether or not a relationship-type can be defined by a user of which this entity-type is a member.

ALTERNATE-ENTITY-TYPE-NAME

An alternate name for the entity-type which is unique in the schema.

ENTITY-CLASS

This meta-attribute-type specifies whether the entity-type represents DATA, PROCESS, EXTERNAL, or SECURITY entities. This meta-attribute-type is used in the definition of some relationship-class-types in the system-standard schema.

The following meta-attribute-types apply to all meta-entities of the type **RELATIONSHIP-TYPE**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not

modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

SYSTEM-LOCK

An indicator, which, when it has the value ON, will disallow any attempt to delete this meta-entity.

INSTALLATION-LOCK

Similar to SYSTEM-LOCK, but special commands are available to override this lock.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

INVERSE-NAME

An optional meta-attribute-type that provides a name for the relationship-type that allows a relationship to be specified in the inverse order.

STATUS-RELATED

Specifies whether or not the relationship-type is used in the integrity rules for the "CONTROLLED" status, which is discussed in Section 3.1.5. The default value is NO.

SEQUENCED

Specifies whether or not the instances of the relationship-type are sequenced. The default value is NO.

SEQUENCE-PARAMETER

Specifies, in the case where instances are sequenced, the manner in which they are to be sequenced. It is required in that case.

The following meta-attribute-types apply to all meta-entities of type **RELATIONSHIP-CLASS-TYPE**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

SYSTEM-LOCK

An indicator, which, when it has the value ON, will disallow any attempt to delete this meta-entity.

INSTALLATION-LOCK

Similar to SYSTEM-LOCK, but special commands are available to override this lock.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

INVERSE-NAME

An optional meta-attribute-type that provides a name allowing for the relationship-group to be specified in the inverse order.

The following meta-attribute-types apply to all meta-entities of type **ATTRIBUTE-TYPE**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

SYSTEM-LOCK

An indicator, which, when it has the value ON, will

disallow any attempt to delete this meta-entity.

INSTALLATION-LOCK

Similar to SYSTEM-LOCK, but special commands are available to override this lock.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

MINIMUM-LENGTH

Specifies the minimum number of characters which an attribute of the given type must have.

MAXIMUM-LENGTH

Same as above, but for maximum length.

PICTURE

The picture of an attribute of the given type. If multiple pictures are used, the attribute must be one of the representations given.

ALTERNATE-ATTRIBUTE-TYPE-NAME

As for entity-types, allows an alternate unique name to be used for the attribute-type.

SYSTEM-GENERATED

Specifies that the attributes of the given type are generated by the system. In this case a value of the meta-attribute-type ID-START must be given; the first attribute generated will be this value and subsequent attributes will be formed by incrementing the preceding name by 1.

ID-START

The starting value for system-generated attributes. It must be a string of the form A...N..., where A denotes an alphanumeric character and N an integer. If the system-generated attributes option is used, any values specified for the meta-attribute-types MINIMUM-LENGTH, MAXIMUM-LENGTH, and PICTURE are ignored.

The following meta-attribute-types apply to all meta-entities of type **ATTRIBUTE-GROUP-TYPE**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

SYSTEM-LOCK

An indicator, which, when it has the value ON, will

disallow any attempt to delete this meta-entity.

INSTALLATION-LOCK

Similar to SYSTEM-LOCK, but special commands are available to override this lock.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

The following meta-attribute-type applies to all meta-entities of type **ATTRIBUTE-TYPE-VALIDATION-PROCEDURE**:

PURPOSE

A string of text intended to be used for a description of the meta-entity and its purpose.

No other meta-attribute-types exist for meta-entities of this type, as no facilities exist in the DDS core standard for modifying existing meta-entities of this type or creating new ones.

The following meta-attribute-types apply to all meta-entities of type **ATTRIBUTE-TYPE-VALIDATION-DATA**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

VALUE/RANGE

- (a) If VALUE is specified as the meta-attribute, the meta-attribute-type DATA-VALUE will contain discrete values against which validation will take place.
- (b) If RANGE is specified, the meta-attribute-type DATA-RANGE will contain ranges against which validation will occur.

There is no default for this meta-attribute-type.

DATA-VALUE

The meta-attributes of this meta-attribute-type only apply when VALUE/RANGE has been specified as VALUE. In that case the meta-attributes represent the allowable values for the attribute-type that is connected to the ATTRIBUTE-TYPE-VALIDATION-DATA meta-entity. In addition to the allowable value, a meta-attribute may also optionally contain a string of literals that can, for example, provide the explanation of the value specified. The maximum length of the string is an implementor option.

As an example, suppose that an attribute-type-validation-data meta-attribute is used to store a list of allowable codes, and that, for instance 29 is the code that is used for the state-name Ohio. This could be expressed by the clause

DATA-VALUE = 29 "OHIO".

DATA-RANGE

The meta-attributes of this meta-attribute-type only apply when VALUE/RANGE has been specified as RANGE. In that case the meta-attributes represent the allowable ranges for the attribute-type that is connected to the ATTRIBUTE-TYPE-VALIDATION-DATA meta-entity. In addition to the allowable range a meta-attribute may also optionally contain a string of literals that can, for example, provide the explanation of the range specified. The maximum length of the string is an implementor option.

The following meta-attribute-types apply to all meta-entities of type **STATUS-NAME**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by

a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

STATUS-LOCK

An indicator which, when set to the value YES, will disallow this meta-entity to be deleted.

CONTROLLED/UNCONTROLLED

An indicator which serves to identify whether the STATUS-NAME meta-entity represents a controlled or uncontrolled status (the definition of these terms being given in Section 3.1.5). Exactly one controlled status exists in the schema. The default for this

meta-attribute is UNCONTROLLED.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

The following meta-attribute-types apply to all meta-entities of type **STAGE-NAME**:

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

DATE-LAST-MODIFIED-IN-SCHEMA

Date and time of the last modification of the meta-entity in the schema; generated by the system; is not modifiable by a user.

LAST-MODIFIED-IN-SCHEMA-BY

Identification of the user responsible for the latest

modification of the meta-entity in the schema; generated by the system; cannot be modified by a user.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA

The number of modifications of a meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

3.1.4 META-ATTRIBUTE-TYPES OF META-RELATIONSHIP-TYPES

The following meta-attribute-types apply to the meta-relationship-type

M-R-T(RELATIONSHIP-TYPE, ENTITY-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

POSITION

Meta-attributes of this type are either FIRST or SECOND, and are used to designate whether the entity-type in the meta-relationship-type is the first or second member of the relationship-type specified. This meta-attribute-type is required for every such meta-relationship-type.

For example, if a relationship-type named WROTE existed with members PROGRAMMER and PROGRAM (in that order), then the meta-relationship-type

M-R-T(WROTE, PROGRAMMER)

would have the meta-attribute-type POSITION with value FIRST, and the meta-relationship-type

M-R-T(WROTE, PROGRAM)

would have the value SECOND for this meta-attribute-type.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T(ENTITY-TYPE, ATTRIBUTE-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the

schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

SINGULAR/PLURAL

If the value SINGULAR is specified, every entity of the type given in the command can have at most one attribute of the type stated in the command. If the value PLURAL is specified, multiple attributes may occur for a single entity.

MAXIMUM-NUMBER-OF-OCCURRENCES

This meta-attribute-type is only valid when the specification for SINGULAR/PLURAL is PLURAL, and specifies the maximum number of occurrences of attributes for a single entity. The default value is an implementor-defined maximum value for the system.

USE-AS-IDENTIFIER

If the value YES is specified, the designated attribute-type can be used as an identifier for entities of the entity-type stated in the meta-relationship. The system will assure that these attributes are unique within the dictionary, as is the case with the primary name of an entity. As will be discussed

in Section 6.1.2, such attributes can be used in certain commands in place of the primary name of an entity.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T (RELATIONSHIP-TYPE, ATTRIBUTE-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

SINGULAR/PLURAL

If the value SINGULAR is specified, every entity of the type given in the command can have at most one attribute of the type stated in the command. If the value PLURAL is specified, multiple attributes may occur for a single entity.

MAXIMUM-NUMBER-OF-OCCURRENCES

This meta-attribute-type is only valid when the specification for SINGULAR/PLURAL is PLURAL, and specifies the maximum number of occurrences of attributes for a single entity. The default value is an implementor-defined maximum value for the system.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T (RELATIONSHIP-CLASS-TYPE, RELATIONSHIP-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T (ATTRIBUTE-GROUP-TYPE, ATTRIBUTE-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

GROUP-POSITION

Served to specify the position that the attribute-type has in the ordered set that constitutes the attribute-group-type. Allowable attributes are FIRST, SECOND, This meta-attribute is required for every meta-relationship-type of this kind.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T (ENTITY-TYPE, ATTRIBUTE-GROUP-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

SINGULAR/PLURAL

If the value SINGULAR is specified, every entity of the type given in the command can have at most one attribute-group of the type stated in the command. If the value PLURAL is specified, multiple attribute-groups may occur for a single entity.

MAXIMUM-NUMBER-OF-OCCURRENCES

This meta-attribute-type is only valid when the specification for SINGULAR/PLURAL is PLURAL, and specifies the maximum number of occurrences of attribute-groups for a single entity. The default value is an implementor-defined maximum value for the system.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T(RELATIONSHIP-TYPE, ATTRIBUTE-GROUP-TYPE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

SINGULAR/PLURAL

If the value SINGULAR is specified, every relationship of the type given in the command can have at most one attribute-group of the type stated in the command. If the value PLURAL is specified, multiple attribute-groups may occur for a single relationship.

MAXIMUM-NUMBER-OF-OCCURRENCES

This meta-attribute-type is only valid when the specification for SINGULAR/PLURAL is PLURAL, and specifies the maximum number of occurrences of attribute-groups for a single relationship. The default value is an implementor-defined maximum value for the system.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T (ATTRIBUTE-TYPE, ATTRIBUTE-TYPE-VALIDATION-PROCEDURE)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

The following meta-attribute-types apply to the meta-relationship-type

M-R-T (ATTRIBUTE-TYPE, ATTRIBUTE-TYPE-VALIDATION-DATA)

DATE-CREATED-IN-SCHEMA

Date and time of creation of the meta-entity in the schema; generated by the system; is not modifiable by a user.

CREATED-IN-SCHEMA-BY

Identification of the dictionary user responsible for the creation of the meta-entity in the schema; generated by the system; cannot be modified by a user.

BASIC/EXTENDED

An indicator that shows whether the meta-entity is part of the system-standard schema or was created by the installation.

PURPOSE

A user-supplied string of text intended to be used for a description of the meta-entity and its purpose.

3.1.5 THE STATUS FACILITY OF THE DICTIONARY SCHEMA

The dictionary schema recognizes two different kinds of STATUS-NAME meta-entities:

- o There exists a single STATUS-NAME for which the meta-attribute-type CONTROLLED/UNCONTROLLED has the value CONTROLLED.
- o There exist one or more STATUS-NAME meta-entities for which the meta-attribute-type CONTROLLED/UNCONTROLLED has the value UNCONTROLLED.

These meta-entities do not participate in any meta-relationships and have meta-attributes of the meta-attribute-types previously specified.

The semantics of these statuses is described in detail as part of the commands specified in Chapter 6, and can be characterized as follows:

An entity always is in one status.

When an entity is added to the dictionary it must be in an UNCONTROLLED status. One such status is designated for every user as being the default status for such new entities being added.

When an entity is in the CONTROLLED status it may not be modified or deleted.

Structural integrity of the CONTROLLED status is enforced by the system based on relationships that are instances of relationship-types which are members of the relationship-class-types CONTAINS and PROCESSES. For this purpose the following hierarchy (from highest to lowest) of entity-types is used:

SYSTEM
PROGRAM
MODULE
FILE
DOCUMENT
RECORD
ELEMENT

No facilities exist in the core standard for entity-types that are created through extensibility facilities to participate in this hierarchy.

Examples of the structural integrity of the CONTROLLED status are given in the discussion of the CHANGE-STATUS command.

3.1.6 THE STAGE FACILITY OF THE DICTIONARY SCHEMA

The system-standard schema does not contain any STAGE-NAME meta-entities and it is expected that each installation will create such schema descriptors with names that agree with the system life cycle methodology in use at the installation. The names assigned to these schema descriptors will be the attributes of the STAGE attribute-type.

These meta-entities do not participate in any meta-relationships and have the meta-attribute-types already specified.

3.2 THE DICTIONARY SCHEMA

The Dictionary Schema consists of:

- a) a set of entity-types
- b) a set of relationship-types
- c) a set of attribute-types
- d) a set of relationship-class-types
- e) a set of attribute-group-types
- f) a set of ATTRIBUTE-TYPE-VALIDATION-PROCEDURES
- g) a set of ATTRIBUTE-TYPE-VALIDATION-DATA
- h) a set of STATUS-NAMES
- i) a set of STAGE-NAMES

The usage of the term Schema Descriptor is extended to include any member of one of the sets a) through i).

Any member of one of the sets f) through i) will be referred to as a Control Descriptor of the Schema. As the name implies, the control descriptors contain information that is used in the control and integrity features for the dictionary and the schema as well as the operation of the DDS.

The schema also contains instances of the meta-relationship-types specified in 3.1.2, as well as instances of the meta-attribute-types specified in 3.1.3 and 3.1.4.

3.3 CUSTOMIZATION OF THE DICTIONARY SCHEMA

The core standard contains the specification of a System-Standard Schema, as given in Chapter 4, which consists of a specific set of entity-types, relationship-types, and attribute-types, which are supported by the DDS core standard. There exists a recognized need to provide facilities that allow an installation to augment this system-standard schema. To this effect the core standard contains facilities that allow an installation to define new schema descriptors and to connect them with meta-relationships, and to assign meta-attributes to both of these. The facilities required for this purpose will be referred to as the Extensibility Facility of the DDS. The commands available to invoke these facilities are specified in Chapter 5. Commands are also specified that allow modifications (i.e., alterations and deletions) of the dictionary schema, as well as provide for query and reporting on the contents of the dictionary schema.

The domain of usage of the extensibility facility includes the system-standard schema with the following exception:

- o Certain descriptors in the system-standard schema are identified with the value ON for the meta-attribute-type SYSTEM-LOCK. The existence of these descriptors is required for certain parts of the functionality of the DDS, and as such, no commands are provided for their deletion.
- o Certain descriptors in the system-standard schema are identified with the value ON for the meta-attribute-type INSTALLATION-LOCK. Whereas the existence of these descriptors is not essential for the operation of the DDS, it is unlikely that they would ever be deleted by an installation. Normal maintenance commands are inhibited from operating on such descriptors, and a special set of such commands are provided for them.

A potential objection to the inclusion of extensibility facilities in the core standard was that this specification should not be limited to implementations on relatively large

computer systems. Appendix A, which is not considered to be part of the specification of the core standard, presents a discussion showing how these facilities can be implemented with a relational DBMS, such a system being commonly available on even very small computers. The implementation method presented is not to be considered to be mandatory in the standard, but is only presented as a verification of feasibility of achieving such an objective.

3.4 THE DICTIONARY

The Dictionary consists of

- a) a set of entities
- b) a set of relationships
- c) a set of attributes

such that:

1. Every entity has exactly one entity-type, which exists in the schema.
2. Every relationship has exactly one relationship-type, which exists in the schema.
3. Every attribute has exactly one attribute-type, which exists in the schema.
4. The dictionary is subject to the integrity rules for statuses specified in the schema.
5. The attributes in the dictionary are subject to the attribute-type-validation-procedures and attribute-type-validation-data in the schema.
6. Every entity in the dictionary has a name, which is called the **primary name** of the entity. The primary name of an entity is unique in the dictionary.

7. The primary name of an entity is composed of two parts which are concatenated to form the primary name. The first part is the **assigned name** of the entity and the second part is the **version number**. The default for version number is the integer "1".
8. The assigned name of an entity is specified when the entity is created in the dictionary through the use of either an ADD-ENTITY, COPY, or RENAME command (which are discussed in Chapter 6). This name either
 - a) is assigned by the user, or
 - b) is generated by the dictionary system for every entity whose entity-type has been designated as having system-generated primary names.

The assigned name specified in one of these commands cannot be the assigned name of an entity existing in the dictionary.

9. Entities for which the primary name is generated by the system are instances of an entity-type for which the meta-attribute-type SYSTEM-GENERATED has the value YES, and the required ID-START meta-attribute has been specified. Such an entity-type will be referred to as having **system-generated assigned names**. Rules governing such entity-types will be discussed in the rules for the commands for interaction with the dictionary.
10. The assigned name of an entity is subject to naming rules existing in the schema.

CHAPTER 4. THE SYSTEM STANDARD SCHEMA

In this chapter we will specify the schema descriptors that are contained in the system-standard schema, along with the applicable meta-attributes that exist for them.

There exist certain meta-attribute-types which are assigned to every descriptor in the system-standard schema. These are to show:

- o which descriptors were included in the system-standard schema versus having been added during a customization step;
- o audit-oriented data on the creation and modification of schema descriptors;
- o control features associated with a meta-entity.

The order of presentation in this section will be the following:

- o We will first specify the entity-types that exist in the system-standard schema with the meta-attributes that exist for them (Section 4.1).
- o This will be followed by the specification of the relationship-types and their meta-attributes. (Section 4.2).
- o Next we will specify the relationship-class-types that exist in the system-standard schema, and also specify the relationship-types of which they are composed (Section 4.3).
- o We will then specify the attribute-types that exist in the system-standard schema and their meta-attributes. We will defer at this point the specifica-

tion of which entity-types and/or relationship-types these apply to (Section 4.4).

- o This will be followed by the specification of the attribute-group-types, showing which attribute-types are included in them (Section 4.5).
- o Next the specification of attribute-type-validation-procedures will be given (Section 4.6).
- o This will be followed by the specification of the attribute-type-validation-data meta-entities in the system-standard schema (Section 4.7).
- o We will then discuss the STATUS-NAME meta-entities in the system-standard schema (Section 4.8).
- o This will be followed by the specification of STAGE-NAME meta-entities in the system-standard schema (Section 4.9).
- o The last item will be a listing of all entity-types and relationship-types showing the attribute-types that are associated with them (Sections 4.10 and 4.11).

4.1 ENTITY-TYPES

The following entity-types, with their associated meta-attributes, exist in the system-standard schema:

1. ENTITY-TYPE NAME = **SYSTEM**

PURPOSE = "To describe instances of collections of processes and data."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = PROCESS

2. ENTITY-TYPE NAME = **PROGRAM**

PURPOSE = "To describe instances of automated processes."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA
will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY
will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = PROCESS

3. ENTITY-TYPE NAME = **MODULE**

PURPOSE = "To describe instances of automated processes which are either logical subdivisions of PROGRAM entities or independent processes which are called by PROGRAM entities."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

DATE-CREATED-IN-SCHEMA
will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY
will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA
will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY
will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = PROCESS

4. ENTITY-TYPE NAME = FILE

PURPOSE = To describe instances of an organization's data collections."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY .

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = DATA

5. ENTITY-TYPE NAME = RECORD

PURPOSE = "To describe instances of logically associated data which belongs to the organization."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = DATA

6. ENTITY-TYPE NAME = DOCUMENT

PURPOSE = "To describe instances of human-readable data collections."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = DATA

7. ENTITY-TYPE NAME = ELEMENT

PURPOSE = "To describe instances of data belonging to the organization."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = DATA

8. ENTITY-TYPE NAME = USER

PURPOSE = "To describe instances of members or collections of members belonging to the organization using the facilities available in the data dictionary system. This entity must not be confused with the entity-type DICTIONARY-USER, which denotes persons who access the dictionary system."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = YES

ENTITY-CLASS = EXTERNAL

9. ENTITY-TYPE NAME = DICTIONARY-USER

PURPOSE = "To describe individuals who are users of the data dictionary system and to record their access privileges to the dictionary. Entities of this type are used exclusively in the management of the security facility of the dictionary system, and are not available through its generally available facilities."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = NO

ENTITY-CLASS = SECURITY

10. ENTITY-TYPE NAME = ACCESS-CONTROLLER

PURPOSE = "To specify access restrictions to an entity or set of entities in the dictionary."

Entities of this type are used exclusively in the security facility of the dictionary system."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = NO

ENTITY-TYPE = SECURITY

The purpose of the entity-types DICTIONARY-USER and ACCESS-CONTROLLER will be explained in Chapter 8 where the security/access control facility for the dictionary and dictionary schema will be specified.

Notes:

1. The meta-attribute for SYSTEM-LOCK is YES in the case of all entity-types whose value for the meta-attribute-type ENTITY-CLASS is DATA. Commands exist in the core standard that depend on the existence of these entity-types in the schema.

2. Meta-attributes for meta-attribute-types

INSTALLATION-LOCK
SYSTEM-GENERATED
META-KEY
MINIMUM-NAME-LENGTH
MAXIMUM-NAME-LENGTH
PICTURE

are not specified. These meta-attributes can be declared by an installation, if so desired. The commands specified in Chapter 5 are to be used for these declarations.

3. The meta-attribute-type CONNECTABLE has the value NO for the meta-entity-types DICTIONARY-USER and ACCESS-CONTROLLER to indicate that no relationship-type with one of these entity-types as a member can be created by a user. The meta-entity-type ACCESS-CONTROLLER is a member of a number of relationship-types, however, these are either a part of the system-standard schema or are established by the system as part of the creation of a new entity-type in the schema.

4.2 RELATIONSHIP-TYPES

The following relationship-types, and their associated meta-attributes, exist in the system-standard schema. The entity-types that are members of the relationship-types are also given. A summary of these relationship-types, showing the entity-types that are members is given in the Entity-type - Relationship-type matrix in Table 4-1. The following notation is used:

1 - denotes the first member of the relationship-type,

2 - denotes the second member of the relationship-type,
and

R - denotes the the same entity-type is both the first and
second member of the relationship-type.

The relationship-class-type (as specified in Section 4.3) to which a relationship-type belongs, is shown as a heading in the rows of the matrix.

ENTITY-TYPE - RELATIONSHIP-TYPE MATRIX

	SYS	PRO	MOD	FIL	DOC	REC	ELE	USR
CONTAINS								
FILE-CONTAINS-FILE	.	.	.	R
FILE-CONTAINS-RECORD	.	.	.	1	.	2	.	.
FILE-CONTAINS-ELEMENT	.	.	.	1	.	.	2	.
RECORD-CONTAINS-RECORD	R	.	.
RECORD-CONTAINS-ELEMENT	1	2	.
ELEMENT-CONTAINS-ELEMENT	R	.
DOCUMENT-CONTAINS-DOCUMENT	R	.	.	.
DOCUMENT-CONTAINS-RECORD	1	2	.	.
DOCUMENT-CONTAINS-ELEMENT	1	.	2	.
SYSTEM-CONTAINS-SYSTEM	R
SYSTEM-CONTAINS-PROGRAM	1	2
SYSTEM-CONTAINS-MODULE	1	.	2
PROGRAM-CONTAINS-PROGRAM	.	R
PROGRAM-CONTAINS-MODULE	.	1	2
MODULE-CONTAINS-MODULE	.	.	R

Table 4-1 (Part 1)

PROCESSES

SYSTEM-PROCESSES-ELEMENT	1	2	.
PROGRAM-PROCESSES-ELEMENT	.	1	2	.
MODULE-PROCESSES-ELEMENT	.	.	1	.	.	.	2	.
SYSTEM-PROCESSES-ELEMENT	1	2	.	.
PROGRAM-PROCESSES-RECORD	.	1	.	.	.	2	.	.
MODULE-PROCESSES-RECORD	.	.	1	.	.	2	.	.
SYSTEM-PROCESSES-FILE	1	.	.	2
PROGRAM-PROCESSES-FILE	.	1	.	2
MODULE-PROCESSES-FILE	.	.	1	2
SYSTEM-PROCESSES-DOCUMENT	1	.	.	.	2	.	.	.
PROGRAM-PROCESSES-DOCUMENT	.	1	.	.	2	.	.	.
MODULE-PROCESSES-DOCUMENT	.	.	1	.	2	.	.	.

RESPONSIBLE-FOR

USER-RESPONSIBLE-FOR-SYSTEM	2	1
USER-RESPONSIBLE-FOR-PROGRAM	.	2	1
USER-RESPONSIBLE-FOR-MODULE	.	.	2	1

Table 4-1 (Part 2)

	SYS	PRO	MOD	FIL	DOC	REC	ELE	USR
USER-RESPONSIBLE-FOR-FILE	.	.	.	2	.	.	.	1
USER-RESPONSIBLE-FOR-DOCUMENT	2	.	.	1
USER-RESPONSIBLE-FOR-RECORD	2	.	1
USER-RESPONSIBLE-FOR-ELEMENT	2	1
RUNS								
USER-RUNS-SYSTEM	2	1
USER-RUNS-PROGRAM	.	2	1
USER-RUNS-MODULE	.	.	2	1
TO								
SYSTEM-TO-SYSTEM	R
PROGRAM-TO-PROGRAM	.	R
MODULE-TO-MODULE	.	.	R
DERIVED-FROM								
ELEMENT-DERIVED-FROM-ELEMENT	R	.
FILE-DERIVED-FROM-FILE	.	.	.	R

Table 4-1 (Part 3)

	SYS	PRO	MOD	FIL	DOC	REC	ELE	USR
DOCUMENT-DERIVED-FROM-DOCUMENT	R	.	.	.
FILE-DERIVED-FROM-DOCUMENT	.	.	.	1	2	.	.	.
DOCUMENT-DERIVED-FROM-FILE	.	.	.	2	1	.	.	.
RECORD-DERIVED-FROM-DOCUMENT	2	1	.	.
DOCUMENT-DERIVED-FROM-RECORD	1	2	.	.
STANDARD-FOR	R	.
HAS-SORT-KEY	.	.	.	1	.	.	2	.
HAS-ACCESS-KEY	.	.	.	1	.	.	2	.

Table 4-1 (Part 4)

1. RELATIONSHIP-TYPE NAME = FILE-CONTAINS-FILE

PURPOSE = "The FILE-CONTAINS-FILE relationship-type is intended to be used to describe instances of a file conceptually containing other files."

INVERSE-NAME = FILE-CONTAINED-IN-FILE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: FILE

Second member: FILE

2. RELATIONSHIP-TYPE NAME = FILE-CONTAINS-RECORD

PURPOSE = "The FILE-CONTAINS-RECORD relationship-type is intended to describe the records that constitute a file."

INVERSE-NAME = RECORD-CONTAINED-IN-FILE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: FILE

Second member: RECORD

3. RELATIONSHIP-TYPE NAME = FILE-CONTAINS-ELEMENT

PURPOSE = "The FILE-CONTAINS-ELEMENT relationship-type is intended to describe the inclusion of an element in a file."

INVERSE-NAME = ELEMENT-CONTAINED-IN-FILE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: FILE

Second member: ELEMENT

4. RELATIONSHIP-TYPE NAME = RECORD-CONTAINS-RECORD

PURPOSE = "The RECORD-CONTAINS-RECORD relationship-type is intended to describe the inclusion of a record in another record."

INVERSE-NAME = RECORD-CONTAINED-IN-RECORD

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: RECORD

Second member: RECORD

5. RELATIONSHIP-TYPE NAME = RECORD-CONTAINS-ELEMENT

PURPOSE = "The RECORD-CONTAINS-ELEMENT relationship-type is intended to describe the inclusion of an element in a record."

INVERSE-NAME = ELEMENT-CONTAINED-IN-RECORD

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

SEQUENCED = YES

SEQUENCE-PARAMETER = INTEGER

First member: RECORD

Second member: ELEMENT

6. RELATIONSHIP-TYPE NAME = ELEMENT-CONTAINS-ELEMENT

PURPOSE = "The ELEMENT-CONTAINS-ELEMENT relationship-type is intended to be used to describe that an element is composed of other elements."

INVERSE-NAME = ELEMENT-CONTAINED-IN-ELEMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: ELEMENT

Second member: ELEMENT

7. RELATIONSHIP-TYPE NAME = DOCUMENT-CONTAINS-DOCUMENT

PURPOSE = "The DOCUMENT-CONTAINS-DOCUMENT relationship-type is intended to describe instances of documents being contained in other documents."

INVERSE-NAME = DOCUMENT-CONTAINED-IN-DOCUMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: DOCUMENT

Second member: DOCUMENT

8. RELATIONSHIP-TYPE NAME = DOCUMENT-CONTAINS-RECORD

PURPOSE = "The DOCUMENT-CONTAINS-RECORD relationship-type is intended to describe instances of records being contained in a document."

INVERSE-NAME = RECORD-CONTAINED-IN-DOCUMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

First member: DOCUMENT

Second member: RECORD

9. RELATIONSHIP-TYPE NAME = DOCUMENT-CONTAINS-ELEMENT

PURPOSE = "The DOCUMENT-CONTAINS-ELEMENT relationship-type is intended to be used to describe instances of elements being contained in a document."

INVERSE-NAME = ELEMENT-CONTAINED-IN-DOCUMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

SEQUENCED = YES

SEQUENCE-PARAMETER = INTEGER

First member: DOCUMENT

Second member: ELEMENT

10. RELATIONSHIP-TYPE NAME = SYSTEM-CONTAINS-SYSTEM

PURPOSE = "The SYSTEM-CONTAINS-SYSTEM relationship-type is intended to be used to describe that one system is composed conceptually of other systems."

INVERSE-NAME = SYSTEM-CONTAINED-IN-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: SYSTEM

Second member: SYSTEM

11. RELATIONSHIP-TYPE NAME = SYSTEM-CONTAINS-PROGRAM

PURPOSE = "The SYSTEM-CONTAINS-PROGRAM relationship-type is intended to be used to describe that a program is considered conceptually to be included in a system."

INVERSE-NAME = PROGRAM-CONTAINED-IN-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: SYSTEM

Second member: PROGRAM

12. RELATIONSHIP-TYPE NAME = SYSTEM-CONTAINS-MODULE

PURPOSE = "The SYSTEM-CONTAINS-MODULE relationship-type is intended to be used to describe that a module is considered conceptually to be included in a system."

INVERSE-NAME = MODULE-CONTAINED-IN-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

SEQUENCED = YES

SEQUENCE-PARAMETER = INTEGER

First member: SYSTEM

Second member: MODULE

13. RELATIONSHIP-TYPE NAME = PROGRAM-CONTAINS-PROGRAM

PURPOSE = "The PROGRAM-CONTAINS-PROGRAM relationship-type is intended to be used to describe a program calling programs."

INVERSE-NAME = PROGRAM-CONTAINED-IN-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

SEQUENCED = YES

SEQUENCE-PARAMETER = INTEGER

First member: PROGRAM

Second member: PROGRAM

14. RELATIONSHIP-TYPE NAME = PROGRAM-CONTAINS-MODULE

PURPOSE = "The PROGRAM-CONTAINS-MODULE relationship-type is intended to be used to describe that a program calls modules."

INVERSE-NAME = MODULE-CONTAINED-IN-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

SEQUENCED = YES

SEQUENCE-PARAMETER = INTEGER

First member: PROGRAM

Second member: MODULE

15. RELATIONSHIP-TYPE NAME = **MODULE-CONTAINS-MODULE**

PURPOSE = "The MODULE-CONTAINS-MODULE relationship-type is intended to be used to describe a module calling modules."

INVERSE-NAME = MODULE-CONTAINED-IN-MODULE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

SEQUENCED = YES

SEQUENCE-PARAMETER = INTEGER

First member: MODULE

Second member: MODULE

16. RELATIONSHIP-TYPE NAME = **SYSTEM-PROCESSES-ELEMENT**

PURPOSE = "The SYSTEM-PROCESSES-ELEMENT relationship-type is intended to represent that a system processes an element."

INVERSE-NAME = ELEMENT-PROCESSED-BY-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: SYSTEM

Second member: ELEMENT

17. RELATIONSHIP-TYPE NAME = **PROGRAM-PROCESSES-ELEMENT**

PURPOSE = "The PROGRAM-PROCESSES-ELEMENT relationship-type is intended to represent that a program processes an element."

INVERSE-NAME = ELEMENT-PROCESSED-BY-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: PROGRAM

Second member: ELEMENT

18. RELATIONSHIP-TYPE NAME = **MODULE-PROCESSES-ELEMENT**

PURPOSE = "The MODULE-PROCESSES-ELEMENT relationship-type is intended to represent that a module processes an element."

INVERSE-NAME = ELEMENT-PROCESSED-BY-MODULE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: MODULE

Second member: ELEMENT

19. RELATIONSHIP-TYPE NAME = **SYSTEM-PROCESSES-RECORD**

PURPOSE = "The SYSTEM-PROCESSES-RECORD relationship-type is intended to represent that a system processes a record."

INVERSE-NAME = RECORD-PROCESSED-BY-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: SYSTEM

Second member: RECORD

20. RELATIONSHIP-TYPE NAME = **PROGRAM-PROCESSES-RECORD**

PURPOSE = "The PROGRAM-PROCESSES-RECORD relationship-type is intended to represent that a program processes a record."

INVERSE-NAME = RECORD-PROCESSED-BY-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: PROGRAM

Second member: RECORD

21. RELATIONSHIP-TYPE NAME = MODULE-PROCESSES-RECORD

PURPOSE = "The MODULE-PROCESSES-RECORD relationship-type is intended to represent that a module processes a record."

INVERSE-NAME = RECORD-PROCESSED-BY-MODULE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: MODULE

Second member: RECORD

22. RELATIONSHIP-TYPE NAME = SYSTEM-PROCESSES-FILE

PURPOSE = "The SYSTEM-PROCESSES-FILE relationship-type is intended to represent that a system processes a file."

INVERSE-NAME = FILE-PROCESSED-BY-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: SYSTEM

Second member: FILE

23. RELATIONSHIP-TYPE NAME = PROGRAM-PROCESSES-FILE

PURPOSE = "The PROGRAM-PROCESSES-FILE relationship-type is intended to represent that a program processes a file."

INVERSE-NAME = FILE-PROCESSED-BY-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: PROGRAM

Second member: FILE

24. RELATIONSHIP-TYPE NAME = MODULE-PROCESSES-FILE

PURPOSE = "The MODULE-PROCESSES-FILE relationship-type is intended to represent that a module processes a file."

INVERSE-NAME = FILE-PROCESSED-BY-MODULE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: MODULE

Second member: FILE

25. RELATIONSHIP-TYPE NAME = SYSTEM-PROCESSES-DOCUMENT

PURPOSE = "The SYSTEM-PROCESSES-DOCUMENT relationship-type is intended to represent that a system processes a document."

INVERSE-NAME = DOCUMENT-PROCESSED-BY-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: SYSTEM

Second member: DOCUMENT

26. RELATIONSHIP-TYPE NAME = PROGRAM-PROCESSES-DOCUMENT

PURPOSE = "The PROGRAM-PROCESSES-DOCUMENT relationship-type is intended to represent that a program processes a document."

INVERSE-NAME = DOCUMENT-PROCESSED-BY-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: PROGRAM

Second member: DOCUMENT

27. RELATIONSHIP-TYPE NAME = MODULE-PROCESSES-DOCUMENT

PURPOSE = "The MODULE-PROCESSES-DOCUMENT relationship-type is intended to represent that a module processes a document."

INVERSE-NAME = DOCUMENT-PROCESSED-BY-MODULE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = YES

First member: MODULE

Second member: DOCUMENT

28. RELATIONSHIP-TYPE NAME = **USER-REPONSIBLE-FOR-SYSTEM**

PURPOSE = "The USER-RESPONSIBLE-FOR-SYSTEM relationship-type is intended to represent that an organizational component has responsibility for a system."

INVERSE-NAME = SYSTEM-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: SYSTEM

29. RELATIONSHIP-TYPE NAME = **USER-REPONSIBLE-FOR-PROGRAM**

PURPOSE = "The USER-RESPONSIBLE-FOR-PROGRAM relationship-type is intended to represent that an organizational component has responsibility for a program."

INVERSE-NAME = PROGRAM-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: PROGRAM

30. RELATIONSHIP-TYPE NAME = USER-REPONSIBLE-FOR-MODULE

PURPOSE = "The USER-RESPONSIBLE-FOR-MODULE relationship-type is intended to represent that an organizational component has responsibility for a module."

INVERSE-NAME = MODULE-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: MODULE

31. RELATIONSHIP-TYPE NAME = USER-REPONSIBLE-FOR-FILE

PURPOSE = "The USER-RESPONSIBLE-FOR-FILE relationship-type is intended to represent that an organizational component has responsibility for a file."

INVERSE-NAME = FILE-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: FILE

32. RELATIONSHIP-TYPE NAME = USER-REPONSIBLE-FOR-DOCUMENT

PURPOSE = "The USER-RESPONSIBLE-FOR-DOCUMENT relationship-type is intended to represent that an organizational component has responsibility for a document."

INVERSE-NAME = DOCUMENT-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: DOCUMENT

33. RELATIONSHIP-TYPE NAME = USER-REPONSIBLE-FOR-RECORD

PURPOSE = "The USER-RESPONSIBLE-FOR-RECORD relationship-type is intended to represent that an organizational component has responsibility for a record."

INVERSE-NAME = RECORD-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: RECORD

34. RELATIONSHIP-TYPE NAME = USER-REPOSNSIBLE-FOR-ELEMENT

PURPOSE = "The USER-RESPONSIBLE-FOR-ELEMENT relationship-type is intended to represent that an organizational component has responsibility for an element."

INVERSE-NAME = ELEMENT-RESPONSIBILITY-OF-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

First member: USER

Second member: ELEMENT

35. RELATIONSHIP-TYPE NAME = USER-RUNS-SYSTEM

PURPOSE = "The USER-RUNS-SYSTEM relationship-type is intended to represent that a person or organizational component is associated with the running of a system."

INVERSE-NAME = SYSTEM-RUN-BY-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: USER

Second member: SYSTEM

36. RELATIONSHIP-TYPE NAME = USER-RUNS-PROGRAM

PURPOSE = "The USER-RUNS-PROGRAM relationship-type is intended to represent that a person or organizational component is associated with the running of a program."

INVERSE-NAME = PROGRAM-RUN-BY-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: USER

Second member: PROGRAM

37. RELATIONSHIP-TYPE NAME = USER-RUNS-MODULE

PURPOSE = "The USER-RUNS-MODULE relationship-type is intended to represent that a person or organizational component is associated with the running of a module."

INVERSE-NAME = MODULE-RUN-BY-USER

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: USER

Second member: MODULE

38. RELATIONSHIP-TYPE NAME = SYSTEM-TO-SYSTEM

PURPOSE = "The SYSTEM-TO-SYSTEM relationship-type is intended to be used to represent a concep-

tual "flow" from one system to another. Instances of this relationship-type identify the "current" and "next" entities representing this "flow".

INVERSE-NAME = SYSTEM-FROM-SYSTEM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: SYSTEM

Second member: SYSTEM

39. RELATIONSHIP-TYPE NAME = **PROGRAM-TO-PROGRAM**

PURPOSE = "The PROGRAM-TO-PROGRAM relationship-type is intended to be used to represent a conceptual "flow" from one program to another. Instances of this relationship-type identify the "current" and "next" entities representing this "flow".

INVERSE-NAME = PROGRAM-FROM-PROGRAM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: PROGRAM

Second member: PROGRAM

40. RELATIONSHIP-TYPE NAME = **MODULE-TO-MODULE**

PURPOSE = "The MODULE-TO-MODULE relationship-type is intended to be used to represent a concep-

tual "flow" from one module to another. Instances of this relationship-type identify the "current" and "next" entities representing this "flow".

INVERSE-NAME = MODULE-FROM-MODULE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: MODULE

Second member: MODULE

41. RELATIONSHIP-TYPE NAME = **ELEMENT-DERIVED-FROM-ELEMENT**

PURPOSE = "The ELEMENT-DERIVED-FROM-ELEMENT relationship-type is intended to be used to describe associations between elements wherein the target element is the result of a calculation involving the source element."

INVERSE-NAME = ELEMENT-PRODUCES-ELEMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: ELEMENT

Second member: ELEMENT

42. RELATIONSHIP-TYPE NAME = **FILE-DERIVED-FROM-FILE**

PURPOSE = "The FILE-DERIVED-FROM-FILE relationship-type is intended to be used to describe associations between files wherein the tar-

get file is the result of calculations involving the source file."

INVERSE-NAME = FILE-PRODUCES-FILE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: FILE

Second member: FILE

43. RELATIONSHIP-TYPE NAME = **DOCUMENT-DERIVED-FROM-DOCUMENT**

PURPOSE = "The DOCUMENT-DERIVED-FROM-DOCUMENT relationship-type is intended to be used to describe associations between documents wherein the target document is the result of a calculation involving the source document."

INVERSE-NAME = DOCUMENT-PRODUCES-DOCUMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: DOCUMENT

Second member: DOCUMENT

44. RELATIONSHIP-TYPE NAME = **FILE-DERIVED-FROM-DOCUMENT**

PURPOSE = "The FILE-DERIVED-FROM-DOCUMENT relationship-type is intended to be used to describe associations between a file and a document

wherein the target file is the result of a calculation involving the source document."

INVERSE-NAME = DOCUMENT-PRODUCES-FILE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: FILE

Second member: DOCUMENT

45. RELATIONSHIP-TYPE NAME = DOCUMENT-DERIVED-FROM-FILE

PURPOSE = "The DOCUMENT-DERIVED-FROM-FILE relationship-type is intended to be used to describe associations between a document and a file wherein the target document is the result of a calculation involving the source file."

INVERSE-NAME = FILE-PRODUCES-DOCUMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: DOCUMENT

Second member: FILE

46. RELATIONSHIP-TYPE NAME = RECORD-DERIVED-FROM-DOCUMENT

PURPOSE = "The RECORD-DERIVED-FROM-DOCUMENT relationship-type is intended to be used to describe associations between a record and a document wherein the target record is the result of a calculation involving the source document."

INVERSE-NAME = DOCUMENT-PRODUCES-RECORD

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: RECORD

Second member: DOCUMENT

47. RELATIONSHIP-TYPE NAME = DOCUMENT-DERIVED-FROM-RECORD

PURPOSE = "The DOCUMENT-DERIVED-FROM-RECORD relationship-type is intended to be used to describe associations between a document and a record wherein the target document is the result of a calculation involving the source record."

INVERSE-NAME = RECORD-PRODUCES-DOCUMENT

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: DOCUMENT

Second member: RECORD

48. RELATIONSHIP-TYPE NAME = STANDARD-FOR

PURPOSE = "The STANDARD-FOR relationship-type is intended to be used to represent conceptual associations between entities of entity-type ELEMENT. There are no system-imposed restrictions on its use, however it is primarily intended to support data element standardization efforts."

INVERSE-NAME = STANDARD-OF

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: ELEMENT

Second member: ELEMENT

49. RELATIONSHIP-TYPE NAME = HAS-SORT-KEY

PURPOSE = "The HAS-SORT-KEY relationship-type, with source entity-type FILE, and target entity-type ELEMENT, is used to designate an element which is used as the sort-key, or as a part of the sort-key of a file."

INVERSE-NAME = IS-SORT-KEY

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: FILE

Second member: ELEMENT

50. RELATIONSHIP-TYPE NAME = HAS-ACCESS-KEY

PURPOSE = "The HAS-ACCESS-KEY relationship-type, with source entity-type FILE, and target entity-type ELEMENT, is used to designate an element which is used as the access-key, or as a part of the access-key of a file."

INVERSE-NAME = IS ACCESS-KEY

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

STATUS-RELATED = NO

First member: FILE

Second member: ELEMENT

In addition to the preceding, the system-standard schema also contains the relationship-types which are used in connection with the local security mechanism, which is discussed in Chapter 8. The first member of each one of these relationship-types is the entity-type ACCESS-CONTROLLER, with the other member being another entity-type (with exception of DICTIONARY-USER).

These relationship-types are:

ACCESS-CONTROLLER-FOR-SYSTEM
ACCESS-CONTROLLER-FOR-PROGRAM
ACCESS-CONTROLLER-FOR-MODULE
ACCESS-CONTROLLER-FOR-FILE
ACCESS-CONTROLLER-FOR-RECORD
ACCESS-CONTROLLER-FOR-DOCUMENT
ACCESS-CONTROLLER-FOR-ELEMENT
ACCESS-CONTROLLER-FOR-USER

For each one of these relationship-types:

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

STATUS-RELATED = NO

4.3 RELATIONSHIP-CLASS-TYPES

The system-standard schema contains the following relationship-class-types with their applicable meta-attributes:

1. RELATIONSHIP-CLASS-TYPE NAME = CONTAINS

PURPOSE = "The CONTAINS relationship-class-type is intended to be used to describe instances of an entity being composed conceptually of other entities. The relationship-class can only be used between entities whose attribute of ENTITY-CLASS is the same."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

INVERSE-NAME = CONTAINED-IN

which is composed of the following relationship-types:

FILE-CONTAINS-FILE
FILE-CONTAINS-RECORD
FILE-CONTAINS-ELEMENT
RECORD-CONTAINS-RECORD
RECORD-CONTAINS-ELEMENT
ELEMENT-CONTAINS-ELEMENT
DOCUMENT-CONTAINS-DOCUMENT
DOCUMENT-CONTAINS-RECORD
DOCUMENT-CONTAINS-ELEMENT
SYSTEM-CONTAINS-SYSTEM
SYSTEM-CONTAINS-PROGRAM
SYSTEM-CONTAINS-MODULE
PROGRAM-CONTAINS-PROGRAM
PROGRAM-CONTAINS-MODULE
MODULE-CONTAINS-MODULE

2. RELATIONSHIP-CLASS-TYPE NAME = PROCESSES

PURPOSE = "The PROCESSES relationship-class-type is intended to be used to represent a conceptual association between entities of type with entity-class PROCESS and entities of type with entity-type DATA. The associations illustrate a PROCESS entity processing a DATA entity."

INVERSE-NAME = PROCESSED-BY

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

which is composed of the following relationship-types:

SYSTEM-PROCESSES-ELEMENT
PROGRAM-PROCESSES-ELEMENT
MODULE-PROCESSES-ELEMENT
SYSTEM-PROCESSES-RECORD
PROGRAM-PROCESSES-RECORD
MODULE-PROCESSES-RECORD
SYSTEM-PROCESSES-FILE
PROGRAM-PROCESSES-FILE
MODULE-PROCESSES-FILE
SYSTEM-PROCESSES-DOCUMENT
PROGRAM-PROCESSES-DOCUMENT
MODULE-PROCESSES-DOCUMENT

3. RELATIONSHIP-CLASS-TYPE NAME = RESPONSIBLE-FOR

PURPOSE = "The RESPONSIBLE-FOR relationship-class-type is intended to be used to represent a conceptual association between entities representing organizational components and other entities to denote organizational responsibility."

INVERSE-NAME = RESPONSIBILITY-OF

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

which is composed of the following relationship-types:

USER-RESPONSIBLE-FOR-SYSTEM
USER-RESPONSIBLE-FOR-PROGRAM
USER-RESPONSIBLE-FOR-MODULE
USER-RESPONSIBLE-FOR-FILE
USER-RESPONSIBLE-FOR-DOCUMENT
USER-RESPONSIBLE-FOR-RECORD
USER-RESPONSIBLE-FOR-ELEMENT

4. RELATIONSHIP-CLASS-TYPE NAME = RUNS

PURPOSE = "The RUNS relationship-class-type is intended to be used to represent a conceptual association between the USER entity-type and entity-types of the entity-class PROCESS. The relationship illustrates that a person or organizational component runs a certain process."

INVERSE-NAME = RUN-BY

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

which is composed of the following relationship-types:

USER-RUNS-SYSTEM
USER-RUNS-PROGRAM
USER-RUNS-MODULE

5. RELATIONSHIP-CLASS-TYPE NAME = **TO**

PURPOSE = "The TO relationship-class-type is intended to be used to represent a conceptual "flow" association between entity-types of entity-class PROCESS. Instances of this relationship-class-type identify the "current" and "next" process entities.

INVERSE-NAME = FROM

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

which is composed of the following relationship-types:

SYSTEM-TO-SYSTEM
PROGRAM-TO-PROGRAM
MODULE-TO-MODULE

6. RELATIONSHIP-CLASS-TYPE NAME = **DERIVED-FROM**

PURPOSE = "The DERIVED-FROM relationship-class-type is intended to be used to describe associations between entities where the target entity is the result of a calculation involving the source entity."

INVERSE-NAME = PRODUCES

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

which is composed of the following relationship-types:

ELEMENT-DERIVED-FROM-ELEMENT
FILE-DERIVED-FROM-FILE
DOCUMENT-DERIVED-FROM-DOCUMENT
FILE-DERIVED-FROM-DOCUMENT

DOCUMENT-DERIVED-FROM-FILE
RECORD-DERIVED-FROM-DOCUMENT
DOCUMENT-DERIVED-FROM-RECORD

4.4 ATTRIBUTE-TYPES

The following attribute-types, with their associated meta-attributes, exist in the system-standard schema:

1. ATTRIBUTE-TYPE NAME = DATE-CREATED

PURPOSE = "The DATE-CREATED attribute-type is used for audit purposes. The attributes are generated by the system, are not modifiable by a user and consist of the time and date an instance of a schema descriptor was created in the dictionary."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

2. ATTRIBUTE-TYPE NAME = CREATED-BY

PURPOSE = "The CREATED-BY attribute-type is used for audit purposes. The attributes are generated by the system, are not modifiable by a user, and consist of the identification of the person responsible for the creation of an instance of a schema descriptor."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

3. ATTRIBUTE-TYPE NAME = LAST-MODIFICATION-DATE

PURPOSE = "The LAST-MODIFICATION-DATE attribute-type is used for audit purposes. The attributes are generated by the system, are not modifiable by a user, and consist of the time and date of the last modification of an instance of a schema descriptor."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

4. ATTRIBUTE-TYPE NAME = LAST-MODIFIED-BY

PURPOSE = "The LAST-MODIFIED-BY attribute-type is used for audit purposes. The attributes are generated by the system, are not modifiable by the user, and consist of the identification of the person responsible for the most recent modification of the instance of a schema descriptor."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

5. ATTRIBUTE-TYPE NAME = NUMBER-OF-MODIFICATIONS

PURPOSE = "The NUMBER-OF-MODIFICATIONS attribute-type is used for audit purposes. The attributes are generated by the system, are not modifiable by a user, and consist of the number of modifications of the instance of a schema descriptor."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

6. ATTRIBUTE-TYPE NAME = **DESCRIPTION**

PURPOSE = "The DESCRIPTION attribute-type is intended to be used to describe or define an instance of a schema descriptor."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

PICTURE = TEXT

7. ATTRIBUTE-TYPE NAME = **COMMENTS**

PURPOSE = "The COMMENTS attribute-type is intended to be used to document clarifying information about an instance of a schema descriptor."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

PICTURE = TEXT

8. ATTRIBUTE-TYPE NAME = **CLASSIFICATION**

PURPOSE = "The CLASSIFICATION attribute-type provides to an installation the capability to specify the use of a classification scheme for entities in the dictionary. The installation can assure that the classification scheme is controlled (i.e. only pre-defined classifications are used) by specifying an attribute-type-validation-data meta-entity for this attribute-type."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

9. ATTRIBUTE-TYPE NAME = **ALTERNATE-NAME**

PURPOSE = "The ALTERNATE-NAME attribute-type is intended to be used to record in the dictionary names, other than the primary name, by which an entity is known. This attribute-type can exist outside the context of the attribute-group-type IDENTIFICATION-NAMES."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

10. ATTRIBUTE-TYPE NAME = **ALTERNATE-NAME-CONTEXT**

PURPOSE = "The ALTERNATE-NAME-CONTEXT attribute-type is intended to be used to document in the dictionary the context or environment in which an alternate name for an entity exists. Typical attributes for this attribute-type will be names of programming languages."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

11. ATTRIBUTE-TYPE NAME = **STATUS**

PURPOSE = "The STATUS attribute-type exists to indicate that an entity is either in an "uncontrolled" or "controlled" status. These attributes are not modifiable by a user in the same manner as other attributes,

but special commands subject to system integrity rules are required to effect a change."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

12. ATTRIBUTE-TYPE NAME = STAGE

PURPOSE = "The STAGE attribute-type exists to indicate the subdivision of the system life cycle in which an entity is considered to be. Attributes for this attribute-type are the names assigned in the STAGE-NAME control elements. No system features are in control to regulate the change from one STAGE to another STAGE."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

13. ATTRIBUTE-TYPE NAME = SECURITY

PURPOSE = "The SECURITY attribute-type exists to document the security requirements of entity instances in the "real world". This attribute-type is not intended to "control" usage of these real world instances; it is only used for documentation."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

EXAMPLES: CLASSIFIED, SECRET, TOP SECRET,
UNCLASSIFIED, SPECIAL

14. ATTRIBUTE-TYPE NAME = **SHORT-NAME**

PURPOSE = "The SHORT-NAME attribute-type, which is associated with the entity-type ELEMENT, can be used in some commands to refer to an entity in place of the primary name of the entity."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

15. ATTRIBUTE-TYPE NAME = **ACCESS-METHOD**

PURPOSE = "The ACCESS-METHOD attribute-type is intended to be used to document the method used by a "real world" program or module to access the data in a "real world" file. Although two programs may use different access methods on the same file, it is not expected that the same program will use two different access methods on the same file. Therefore, this attribute-type will be single-valued."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

EXAMPLES: SEQUENTIAL,RANDOM,ISAM,POINTER

16. ATTRIBUTE-TYPE NAME = **FREQUENCY**

PURPOSE = "The frequency attribute-type is intended to be used to document how often a process occurs. When used as an attribute-type for SYSTEM and PROGRAM, it illustrates the expected frequency of occurrence. When used in

the SYSTEM-CONTAINS-PROGRAM, SYSTEM-CONTAINS-SYSTEM, and PROGRAM-CONTAINS-PROGRAM it illustrates the frequency of occurrence within another process."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

EXAMPLES: WEEKLY, DAILY, YEARLY

17. ATTRIBUTE-TYPE NAME = LOCATION

PURPOSE = "The LOCATION attribute-type exists to document the place(s) within an organization (enterprise) where an entity can be found. The attributes associated with this attribute-type are intended to be human readable/interpretable."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

EXAMPLES: Room 27 Bldg 200; Site A;
1111 Enterprise Ave, North City

18. ATTRIBUTE-TYPE NAME = DURATION-VALUE

PURPOSE = "The DURATION-VALUE attribute-type is intended to be used to document a magnitude of time. It has no meaning without the context provided by the DURATION-TYPE. Attributes of this type are intended to be numeric."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

19. ATTRIBUTE-TYPE NAME = DURATION-TYPE

PURPOSE = "The DURATION-TYPE attribute-type is intended to be used in conjunction with the DURATION-VALUE attribute-type to illustrate the units associated with the numeric value of DURATION-VALUE."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

EXAMPLES: Days, Seconds

20. ATTRIBUTE-TYPE NAME = NUMBER-OF-RECORDS

PURPOSE = "The NUMBER-OF-RECORDS attribute-type is intended to be used to document the number of logical records expected to exist in the "real world" file instance."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

21. ATTRIBUTE-TYPE NAME = NUMBER-OF-LINES-OF-CODE

PURPOSE = "The NUMBER-OF-LINES-OF-CODE attribute-type is intended to be used to document, based on a user's definition, the number of lines of code, of instructions, of statements, etc. associated with a real world program or module."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

22. ATTRIBUTE-TYPE NAME = USAGE-FORMAT

PURPOSE = "The USAGE-FORMAT attribute-type is intended to be used in the creation of data structures for programming languages by providing the format to be associated with the entity instance or the substitute format as determined by the relationship."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

EXAMPLES:

In the COBOL sense,

- o for a FILE entity, the USAGE-FORMAT would contain the FD clause;
- o for a Record entity, the USAGE-FORMAT would contain the default 01 level clause;
- o for an ELEMENT entity, the USAGE-FORMAT would contain the default PICTURE clause;
- o for a FILE-CONTAINS-RECORD relationship, the USAGE-FORMAT contains the substitute 01 level clause for the RECORD entity;
- o for a RECORD-CONTAINS-ELEMENT relationship, the USAGE-FORMAT contains the substitute PICTURE clause for the ELEMENT entity.

23. ATTRIBUTE-TYPE NAME = LANGUAGE

PURPOSE = "The LANGUAGE attribute-type is used to identify the programming language associated with the entity in the case of PROGRAM and MODULE, and with the USAGE-FORMAT in the case of FILE, RECORD, ELEMENT, FILE-CONTAINS-RECORD and RECORD-CONTAINS-ELEMENT."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

EXAMPLES: COBOL, FORTRAN, PL/I

24. ATTRIBUTE-TYPE NAME = REPRESENTATION-NUMBER

PURPOSE = "The REPRESENTATION-NUMBER attribute-type is used to identify the particular representation to be used in the creation of a programming language data structure."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

25. ATTRIBUTE-TYPE NAME = LENGTH

PURPOSE = "The LENGTH attribute-type is intended to be used to document the default/normal/standard length in characters for an element instance. The USAGE-FORMAT could imply a different length."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

26. ATTRIBUTE-TYPE NAME = DATA-CLASS

PURPOSE = "The DATA-CLASS attribute-type is intended to be used to document the general default character of an ELEMENT instance. The USAGE-FORMAT could imply a different data class."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

EXAMPLES: NUMERIC, ALPHA,
ANY-PRINTABLE-CHARACTER

27. ATTRIBUTE-TYPE NAME = CODE-LIST-LOCATION

PURPOSE = "The CODE-LIST-LOCATION attribute-type is intended to be used to record in the dictionary the location(s) at which a list of codes is located. This location is intended to be a location outside the dictionary and cannot be referenced directly by the dictionary system."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

28. ATTRIBUTE-TYPE NAME = ALLOWABLE-VALUE

PURPOSE = "The ALLOWABLE-VALUE attribute-type is intended to be used to record in the diction-

ary the allowable value(s) that can be taken on by a real world element."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

29. ATTRIBUTE-TYPE NAME = **LOW-OF-RANGE**

PURPOSE = "The LOW-OF-RANGE attribute-type is intended to be used to record in the dictionary the low value of a range of values that can be taken on by a real world element."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

30. ATTRIBUTE-TYPE NAME = **HIGH-OF-RANGE**

PURPOSE = "The HIGH-OF-RANGE attribute type is intended to be used to record in the dictionary the high value of a range of values that can be taken on by a real world element."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

31. ATTRIBUTE-TYPE NAME = **SYSTEM-CATEGORY**

PURPOSE = "The SYSTEM-CATEGORY attribute-type is intended to be used to allow an installation to set up in the dictionary a classification scheme for systems."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

32. ATTRIBUTE-TYPE NAME = DOCUMENT-CATEGORY

PURPOSE = "The DOCUMENT-CATEGORY attribute-type is intended to be used to allow an installation to set up in the dictionary a classification scheme for documents."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

33. ATTRIBUTE-TYPE NAME = RECORD-CATEGORY

PURPOSE = "The RECORD-CATEGORY attribute-type is intended to be used to allow an installation to set up in the dictionary a classification scheme for records."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = OFF

34. ATTRIBUTE-TYPE NAME = USAGE-INDICATOR

PURPOSE = "The USAGE-INDICATOR attribute-type is intended to be used to indicate the purpose of the relationship between two entities. Although it can be used to identify user-defined usages, particular values associated with the attribute-type will trigger special

actions in the creation of data structures. The "reserved" attributes are GROUP, REDEFINES and CONDITION.

In the RECORD-CONTAINS-RECORD instances, if REDEFINES is the USAGE-INDICATOR instance, then the target record is the source of the redefinition and its associated usage formats from either RECORD-CONTAINS-ELEMENT instances or ELEMENT instances are used to create the redefinition in the COBOL data structure using "66" level elements.

In the ELEMENT-CONTAINS-ELEMENT instances,

- o if GROUP is used, a group structure is created and the target elements contained (related to) the same source element becomes the elements of the group.
- o if CONDITION is used, the IDENTIFICATION-VALUE attribute-group-type of the target element will be used to provide an alternate-name and associated value to create an "88" clause for the source element."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

35. ATTRIBUTE-TYPE NAME = REL-POSITION

PURPOSE = "The REL-POSITION attribute-type is intended to be used to identify the relative position of an element from the beginning of a real world record. An attribute of this type would typically be the count of the number of characters which precede it in the record plus 1. This attribute-type is associated with the relationship-type that specifies

the elements that are contained in a record."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = NO

SYSTEM-LOCK = ON

EXAMPLES: REL-POSITION = 1 will typically denote that the left-most character of the related element starts in position 1 of the record.

REL-POSITION = 28 will typically denote that the left-most character of the related element starts in position 28 of the record.

Additional attribute-types exist for the security/access control facility for the dictionary and dictionary schema. These will be specified in Chapter 8, where a discussion of this facility will be given.

4.5 ATTRIBUTE-GROUP-TYPES

The following attribute-group-types, with their associated meta-attributes, exist in the system-standard schema:

1. ATTRIBUTE-GROUP-TYPE NAME = IDENTIFICATION-NAMES

PURPOSE = "The IDENTIFICATION-NAMES attribute-group-type is intended to be used to record alternate names for entities along with the context or environment in which these names are used. This context is visualized to be the name of a programming language, though its

use is not limited to that purpose."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

This attribute-group-type is composed of the following attribute-types (in the order shown):

ALTERNATE-NAME
ALTERNATE-NAME-CONTEXT

2. ATTRIBUTE-GROUP-TYPE NAME = DURATION

PURPOSE = "The DURATION attribute-group-type is composed of DURATION-VALUE and DURATION-TYPE attribute-types. This group-type is intended to be used to document how long a process takes from initiation to completion."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

This attribute-group-type is composed of the following attribute-types (in the order shown):

DURATION-VALUE
DURATION-TYPE

EXAMPLES: Examples are in terms of its included attribute-types.
(5,Days); (8,Seconds)

3. ATTRIBUTE-GROUP-TYPE NAME = REPRESENTATION

PURPOSE = "The REPRESENTATION attribute-group-type is intended to be used in the creation of data structures for programming languages. It is composed of three attribute-types; these are

USAGE-FORMAT, LANGUAGE, and REPRESENTATION-
NUMBER.

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

This attribute-group-type is composed of the following
attribute-types (in the order shown):

USAGE-FORMAT
LANGUAGE
REPRESENTATION-NUMBER

EXAMPLES:

FILE:

USAGE-FORMAT = FD PAY-IN LABELS OMITTED
LANGUAGE = COBOL
REPRESENTATION-NUMBER = 1

ELEMENT:

USAGE-FORMAT = PICTURE X(5)
LANGUAGE = COBOL
REPRESENTATION-NUMBER = 3

RECORD-TO-ELEMENT:

USAGE-FORMAT = PIC 99999, VALUE IS 500
LANGUAGE = COBOL
REPRESENTATION-NUMBER = 1

4. ATTRIBUTE-GROUP-TYPE NAME = USAGE-NAMES

PURPOSE - "This attribute-group-type is composed of
attribute-types with name alternate-name-1
and alternate-name-2. The meaning of these
is as follows:

Let R-T denote one of the relationship-types
in the preceding list, and let T-1 and T-2

denote the entity-types involved in the relationship. The attribute-types alternate-name-1 and alternate-name-2, in this case, correspond to alternate names of entities of entity-types T-1 and T-2, respectively.

Let entity-A and entity-B denote entities of entity-types T-1 and T-2, respectively, and suppose that the relationship of type R-T which has entity-A and entity-B as members, has an attribute-group of type USAGE-NAMES composed of the attributes name-A and name-B. The meaning that is attached to this attribute-group is as follows:

In the relationship where entity-A contains entity-B, when entity-A has the alternate name name-A, then entity-B has the alternate name name-B."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

Further discussion of this attribute-group-type and the rules associated with it are found in Section 6.1.5.

5. ATTRIBUTE-GROUP-TYPE = ALLOWABLE-RANGE

PURPOSE = "The ALLOWABLE-RANGE attribute-group-type is intended to be used to record in the dictionary the allowable range(s) of values that can be taken on by a real world element.

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

This attribute-group-type is composed of the following attribute-types (in the order shown):

LOW-OF-RANGE
HIGH-OF-RANGE

4.6 ATTRIBUTE-TYPE-VALIDATION-PROCEDURE

The system-standard schema contains the following two ATTRIBUTE-TYPE-VALIDATION-PROCEDURE meta-entities:

1. ATTRIBUTE-TYPE-VALIDATION-PROCEDURE
NAME = VALUE-VALIDATION

PURPOSE = "The VALUE-VALIDATION attribute-type-validation-procedure is used whenever it is desired to restrict the attributes of a given attribute-type to a predefined set. The use of this procedure for an attribute-type requires:

- a) The existence of an attribute-type-validation-data descriptor in the schema (with VALUE/RANGE = VALUE) containing the predefined set.
- b) The existence of meta-relationships in the schema that associate both this procedure and the validation-data element described in a) to the attribute-type for which the validation is to occur."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

1. ATTRIBUTE-TYPE-VALIDATION-PROCEDURE

NAME = RANGE-VALIDATION

PURPOSE = "The RANGE-VALIDATION attribute-type-validation-procedure is used whenever it is desired to restrict the attributes of a given attribute-type to a predefined set of ranges. The use of this procedure for an attribute-type requires:

- a) The existence of an attribute-type-validation-data descriptor in the schema (with VALUE/RANGE = RANGE) containing the predefined set of ranges.
- b) The existence of meta-relationships in the schema that associate both this procedure and the validation-data element described in a) to the attribute-type for which the validation is to occur."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

4.7 ATTRIBUTE-TYPE-VALIDATION-DATA

The system-standard schema contains the following single ATTRIBUTE-TYPE-VALIDATION-DATA meta-entity:

ATTRIBUTE-TYPE-VALIDATION-DATA
NAME = **STAGE-ATTRIBUTES**

PURPOSE = "The STAGE-ATTRIBUTES attribute-type-validation-data meta-entity contains the names of the stages which are in use. These names are created by creation of STAGE-NAME schema descriptors and are not directly modifiable by a dictionary user. At installation time of the DDS, no values will exist in this meta-entity."

VALUE/RANGE = VALUE

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

It is expected that other attribute-type-validation-data elements will be created by an installation to meet their particular requirements for control of attributes. In particular, the use of the classification schemes will be controlled in this manner.

4.8 STATUS-NAME

The system-standard schema contains the following three STATUS-NAME meta-entities:

1. STATUS-NAME NAME = **CONTROLLED**

PURPOSE = "The status-name with name CONTROLLED is used in connection with the structural integrity controls provided by the system for entities which are used in an operational environment."

BASIC/EXTENDED = BASIC

STATUS-LOCK = ON

CONTROLLED/UNCONTROLLED = CONTROLLED

2. STATUS-NAME NAME = UNCONTROLLED

PURPOSE = "The status-name with name UNCONTROLLED is used for entities that are added to the dictionary."

BASIC/EXTENDED = BASIC

STATUS-LOCK = ON

CONTROLLED/UNCONTROLLED = UNCONTROLLED

DEFAULT-STATUS = YES

3. STATUS-NAME NAME = SECURITY-STATUS

PURPOSE = "The status-name with name SECURITY-STATUS is used for entities of type DICTIONARY-USER and ACCESS-CONTROLLER which are used in the security/access control facility for the dictionary and dictionary schema."

BASIC/EXTENDED = BASIC

STATUS-LOCK = ON

CONTROLLED/UNCONTROLLED = UNCONTROLLED

4.9 STAGE-NAME

The system-standard schema does not contain any meta-entities of type STAGE-NAME.

4.10 ATTRIBUTE-TYPES AND ATTRIBUTE-GROUP-TYPES ASSOCIATED WITH ENTITY-TYPES

In this section the attribute-types and attribute-group-types that are associated with each entity-type will be given. This information is also shown in a graphic form in an Entity-type - Attribute-type matrix in Table 4-2. The notation used in this matrix is the following:

S - denotes that only a single attribute can exist for any entity of that type.

P - denotes that multiple attributes can exist for any entity of that type.

ENTITY-TYPE - ATTRIBUTE-TYPE MATRIX

	SYS	PRO	MOD	FIL	DOC	REC	ELE	USR
DATE-CREATED	S	S	S	S	S	S	S	S
CREATED-BY	S	S	S	S	S	S	S	S
LAST-MODIFICATION-DATE	S	S	S	S	S	S	S	S
LAST-MODIFIED-BY	S	S	S	S	S	S	S	S
NUMBER-OF-MODIFICATIONS	S	S	S	S	S	S	S	S
DESCRIPTION	S	S	S	S	S	S	S	S
COMMENTS	S	S	S	S	S	S	S	S
CLASSIFICATION	P	P	P	P	P	P	P	P
IDENTIFICATION-NAMES	P	P	P	P	P	P	P	P
ALTERNATE-NAME								
ALTERNATE-NAME-CONTEXT								
STATUS	S	S	S	S	S	S	S	S
STAGE	S	S	S	S	S	S	S	S
SECURITY	S	S	S	S	S	S	S	S
SHORT-NAME	S	.
ACCESS-METHOD	.	.	.	S
FREQUENCY	S	S

Table 4-2 (Part 1)

	SYS	PRO	MOD	FIL	DOC	REC	ELE	USR
LOCATION	P	P	P
DURATION	S	S	S
DURATION-VALUE								
DURATION-TYPE								
NUMBER-OF-RECORDS	.	.	.	S
NUMBER-OF-LINES-OF-CODE	.	S	S
REPRESENTATION	.	.	.	P	.	P	P	.
USAGE-FORMAT								
LANGUAGE		S	S					
REPRESENTATION-NUMBER								
LENGTH	S	.
DATA-CLASS	S	.
CODE-LIST-LOCATION	P	.
ALLOWABLE-VALUE	P	.
ALLOWABLE-RANGE	P	.
LOW-OF-RANGE								
HIGH-OF-RANGE								
SYSTEM-CATEGORY	S
DOCUMENT-CATEGORY	S	.	.	.
RECORD-CATEGORY	S	.	.

Table 4-2 (Part 2)

The following attribute-types and attribute-group-types pertain to entities of type **SYSTEM**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]
FREQUENCY	[SINGULAR]
LOCATION	[PLURAL]
SYSTEM-CATEGORY	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
DURATION	[SINGULAR]

The following attribute-types and attribute-group-types pertain to entities of type **PROGRAM**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]
FREQUENCY	[SINGULAR]
LOCATION	[PLURAL]
NUMBER-OF-LINES-OF-CODE	[SINGULAR]
LANGUAGE	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
DURATION	[SINGULAR]

The following attribute-types and attribute-group-types pertain to entities of type **MODULE**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]

STAGE	[SINGULAR]
SECURITY	[SINGULAR]
LOCATION	[PLURAL]
NUMBER-OF-LINES-OF-CODE	[SINGULAR]
LANGUAGE	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
DURATION	[SINGULAR]

The following attribute-types and attribute-group-types pertain to entities of type **FILE**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]
ACCESS-METHOD	[SINGULAR]
NUMBER-OF-RECORDS	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
REPRESENTATION	[PLURAL]

The following attribute-types and attribute-group-types pertain to entities of type **DOCUMENT**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]
DOCUMENT-CATEGORY	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
----------------------	----------

The following attribute-types and attribute-group-types pertain to entities of type **RECORD**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]

NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]
RECORD-CATEGORY	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
REPRESENTATION	[PLURAL]

The following attribute-types and attribute-group-types pertain to entities of type **ELEMENT**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]
LENGTH	[SINGULAR]
DATA-CLASS	[SINGULAR]
CODE-LIST-LOCATION	[PLURAL]
ALLOWABLE-VALUE	[PLURAL]
SHORT-NAME	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
REPRESENTATION	[PLURAL]
ALLOWABLE-RANGE	[PLURAL]

The following attribute-types and attribute-group-types pertain to entities of type **USER**. (The value of the meta-attribute-type SINGULAR/PLURAL, which indicates the number of occurrences permitted for each entity, is given in brackets. This meta-attribute-type is associated with the meta-relationship-type whose members are the given entity-type and attribute-type or attribute-group-type.)

ATTRIBUTE-TYPES:

DATE-CREATED	[SINGULAR]
CREATED-BY	[SINGULAR]
LAST-MODIFICATION-DATE	[SINGULAR]
LAST-MODIFIED-BY	[SINGULAR]
NUMBER-OF-MODIFICATIONS	[SINGULAR]
DESCRIPTION	[SINGULAR]
COMMENTS	[SINGULAR]
CLASSIFICATION	[PLURAL]
STATUS	[SINGULAR]
STAGE	[SINGULAR]
SECURITY	[SINGULAR]

ATTRIBUTE-GROUP-TYPES:

IDENTIFICATION-NAMES	[PLURAL]
----------------------	----------

Attribute-types that pertain to the entity-types

DICTIONARY-USER, and
ACCESS-CONTROLLER

will be specified in Chapter 8, where a discussion of the security/access control facility for both the dictionary and dictionary schema will be given.

4.11 ATTRIBUTE-TYPES AND ATTRIBUTE-GROUP-TYPES ASSOCIATED WITH RELATIONSHIP-TYPES

In this section the attribute-types and attribute-group-types that are associated with relationship-types will be specified.

1. The attribute-type **USAGE-INDICATOR** is associated with the following relationship-types:

RECORD-CONTAINS-RECORD
RECORD-CONTAINS-ELEMENT
ELEMENT-CONTAINS-ELEMENT

Only a single attribute is allowed for any such relationship.

2. The attribute-type **REL-POSITION** is associated with the relationship-type

RECORD-CONTAINS-ELEMENT

Only a single attribute is allowed for any such relationship.

3. The attribute-group-type **REPRESENTATION** is associated with the following relationship-types:

FILE-CONTAINS-RECORD
RECORD-CONTAINS-ELEMENT

Multiple attribute-groups are allowed.

4. The attribute-group-type **IDENTIFICATION-NAMES** is associated with all relationship-types of the relationship-class-type

CONTAINS

Multiple attribute-groups are allowed.

5. The attribute-group-type **USAGE-NAMES**, rules for which are given in Section 6.1.5, is associated with all relationship-types of the relationship-class-type

CONTAINS

Multiple attribute-groups are allowed.

CHAPTER 5. COMMANDS FOR INTERACTION WITH THE DICTIONARY SCHEMA

The commands available for interaction with the dictionary schema fall into two broad categories:

- commands that cause modifications to the schema (Section 5.1), and
- commands that report on the contents of the schema (Section 5.2).

It is assumed in this chapter that an error in any part of a command will cause the entire command not to be executed. Hence, only the actions resulting from a complete execution of a command are specified.

Execution of all commands is subject to the security facility defined in Chapter 8.

5.1 SCHEMA MAINTENANCE COMMANDS

All commands are executable in one of two modes:

- UPDATE: In this mode the command, if error-free, will cause an update to the schema to take place.
- CHECK: In this mode the command will only be checked for errors and no update to the schema will take place.

As discussed in Chapter 2, the default mode is CHECK; the mode is changed by the specification

SCHEMA-MODE = UPDATE

preceding a command, and will remain such for the duration of a session or run unit unless again changed by the specification

SCHEMA-MODE = CHECK

In the specification of each command the following syntactic convention is used:

1. The name of the command
2. The identification of the meta-entity(s) or meta-relationship(s) on which the specified action(s) are to take place.
3. A set (possibly null) of clauses, each one identifying one or more meta-attributes involved in the specified action(s). The order in which these clauses are stated is immaterial. These clauses will be referred to as meta-attribute clauses.
4. A set (possibly null) of parametric clauses, each one of which states an optional feature of the action(s) to be taken. The order in which these clauses are stated is immaterial.
5. *null represents an implementor defined symbol for a null value.

The commands that follow are given in alphabetic order by command name.

5.1.1 ABOLISH-META-ENTITY COMMAND

PURPOSE: To delete a meta-entity and its associated meta-attributes in the schema.

FORMAT: ABOLISH-META-ENTITY
meta-entity-name

RULES:

1. meta-entity-name must exist in the schema as the name of a meta-entity.
2. The meta-attribute-type SYSTEM-LOCK of the meta-entity named must have the value OFF.
3. The meta-attribute-type INSTALLATION-LOCK of the meta-entity named must have the value OFF.
4. The meta-entity named must not be a member of a meta-relationship other than:

- a meta-relationship whose other member is one of the following meta-entities whose type is attribute-type:

DATE-CREATED
CREATED-BY
LAST-MODIFICATION-DATE
LAST-MODIFIED-BY
NUMBER-OF-MODIFICATIONS
DESCRIPTION
COMMENTS
CLASSIFICATION
IDENTIFICATION-NAMES

- a meta-relationship whose other member is the meta-entity ACCESS-CONTROLLER whose type is entity-type.

5. The dictionary must not contain an instance of the meta-entity named.

ACTIONS PERFORMED:

1. The meta-entity named is deleted, along with its associated meta-attributes.
2. The existing meta-relationships whose other member is an attribute-type or entity-type, as stated in Rule 4, are deleted.
3. The command is recorded in the log/audit file.
4. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. meta-entity does not exist in the schema as the name of an entity.
2. The meta-attribute-type SYSTEM-LOCK has the value ON.
3. The meta-attribute-type INSTALLATION-LOCK has the value ON.
4. There exists in the schema a meta-relationship, other than those of Rule 4, of which the named meta-entity is a member.
5. There exists an instance of the named meta-entity in the dictionary.
6. A parametric clause is specified.

EXAMPLE

ABOLISH
LOCATION

Assuming that there exists an entity-type with entity-type-name LOCATION in the schema, this command will delete this entity-type from the schema. No instances of this entity-type can exist in the dictionary in order for this command to execute.

5.1.2 ABOLISH-META-ENTITY-WITH-LOCK COMMAND

PURPOSE: To delete a meta-entity for which the meta-attribute-type INSTALLATION-LOCK has the value ON.

The description of this command is identical to that of the ABOLISH-META-ENTITY command except that Rule 3 and Error Condition 3 do not apply.

5.1.3 ABOLISH-META-RELATIONSHIP COMMAND

PURPOSE: To delete an existing meta-relationship and its associated meta-attributes from the schema.

FORMAT: ABOLISH-META-RELATIONSHIP
meta-entity-name-1 TO meta-entity-name-2

RULES:

1. The meta-entities with names meta-entity-name-1 and meta-entity-name-2 must exist in the schema.
2. The schema must contain a meta-relationship between the given meta-entities.
3. For meta-relationships that affect the dictionary, such a meta-relationship may not be deleted if this deletion affects the information contents of the dictionary.
4. A meta-relationship involving a meta-entity of type attribute-type-validation-data and an attribute-type may not be deleted if there exists in the schema a meta-relationship between this attribute-type and a

meta-entity of type attribute-type-validation-procedure.

ACTIONS PERFORMED:

1. The specified meta-relationship and its associated meta-attributes are deleted from the schema.
2. The command is recorded in the log/audit file.
3. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. There does not exist a meta-entity with name meta-entity-name-1 in the schema.
2. There does not exist a meta-entity with name meta-entity-name-2 in the schema.
3. The specified meta-relationship does not exist in the schema.
4. The deletion of the meta-relationship affects the information contents of the dictionary.
5. The meta-relationship to be deleted has as a member a meta-entity of type attribute-type-validation-data and the attribute-type which is the other member is also the member of a meta-relationship whose other member is a meta-entity of type attribute-type-validation-procedure.
6. A meta-attribute clause is specified.
7. A parametric clause is specified.

EXAMPLES

1. ABOLISH-META-RELATIONSHIP
FILE TO FILE-DATE

This command deletes the meta-relationship (as well as any existing meta-attributes of this meta-relationship) with members FILE and FILE-DATE. Assuming that FILE-DATE is an attribute-type, this command is valid only if there does not exist in the dictionary an instance of a FILE which has an attribute of type FILE-DATE.

2. ABOLISH-META-RELATIONSHIP
FILE-TYPE TO FILE-TYPE-VALUES

It is assumed here that FILE-TYPE is an attribute-type and that FILE-TYPE-VALUES is a meta-entity of type attribute-type-validation-data with VALUE/RANGE = VALUE. In order for such validation to have taken place it was necessary that there also existed in the schema a meta-relationship with members FILE-TYPE and the attribute-type-validation-procedure VALUE-VALIDATION. The command given here will only be allowed to execute after that meta-relationship has been deleted, i.e., the attribute-type FILE-TYPE was no longer connected to an attribute-type-validation-procedure meta-entity.

5.1.4 ALTER-META-ENTITY COMMAND

PURPOSE: To modify one or more meta-attributes of a meta-entity.

FORMAT: ALTER-META-ENTITY
meta-entity-name
clause-1 [,clause-2] ...

RULES:

1. meta-entity-name must exist in the schema as the name of a meta-entity.

2. The meta-attribute clauses are of the form

meta-attribute-type FROM value-1 TO value-2

to indicate the modification that is to take place. A meta-attribute is deleted by specifying

FROM value-1 TO *null

and a meta-attribute is added by specifying

FROM *null TO value-2

Whenever value-1 is not null, this value must exist in the schema.

3. Meta-entities which are not instances of the following meta-entity-types

ENTITY-TYPE
RELATIONSHIP-TYPE
ATTRIBUTE-TYPE
RELATIONSHIP-CLASS-TYPE
ATTRIBUTE-GROUP-TYPE
ATTRIBUTE-TYPE-VALIDATION-DATA
STATUS-NAME
STAGE-NAME

cannot be specified in this command.

4. If the meta-entity represents an attribute-type or attribute-group-type, the meta-attribute-type SYSTEM-LOCK must have the value OFF if an existing meta-attribute is to be modified or deleted.
5. If the meta-entity represents an attribute-type or attribute-group-type, the meta-attribute-type INSTALLATION-LOCK must have the value OFF if an

existing meta-attribute is to be modified or deleted.

6. The following meta-attribute-types must not appear in a meta-attribute clause:

DATE-CREATED-IN-SCHEMA
CREATED-IN-SCHEMA-BY
DATE-LAST-MODIFIED-IN-SCHEMA
LAST-MODIFIED-IN-SCHEMA-BY
NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA
BASIC/EXTENDED
SYSTEM/LOCK
INSTALLATION-LOCK
SYSTEM-GENERATED
STATUS-RELATED

7. Changes to meta-attributes in the list which follows must conform with the dictionary (i.e., cannot create a condition in the schema which contradicts the dictionary).

SINGULAR/PLURAL
MAXIMUM-NUMBER-OF-OCCURRENCES
MINIMUM-NAME-LENGTH
MAXIMUM-NAME-LENGTH
PICTURE
SEQUENCED
MINIMUM-LENGTH
MAXIMUM-LENGTH
VALUE-RANGE
CONNECTABILITY

ACTIONS PERFORMED:

1. The schema is modified according to the meta-attribute-clauses specified.
2. For the given meta-entity:
 - (a) The current date and time is assigned to DATE-LAST-MODIFIED-IN-SCHEMA.
 - (b) the identification of the person submitting

the command is assigned to LAST-MODIFIED-IN-SCHEMA-BY, and

(c) the value of NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA is incremented by 1.

3. The command is recorded in the log/audit file.
4. Notification of the execution of the command is given to the user.

ERROR CONDITIONS:

1. The meta-entity specified does not exist in the schema.
2. A meta-attribute specified to be altered does not exist in the schema.
3. The meta-entity specified is an instance of a meta-entity-type to which the command does not apply (i.e., STATUS-NAME or ATTRIBUTE-TYPE-VALIDATION-PROCEDURE).
4. A meta-attribute clause is given which is not in the required format.
5. The meta-entity given represents an attribute-type or attribute-group-type and SYSTEM-LOCK has the value ON.
6. The meta-entity given represents an attribute-type or attribute-group-type and INSTALLATION-LOCK has the value ON.
7. A meta-attribute clause contains a meta-attribute-type for which the command does not apply.
8. A meta-attribute clause contains a meta-attribute which does not conform with the dictionary.
9. A parametric clause is specified.

EXAMPLES

1. ALTER-META-ENTITY-TYPE
ELEMENT
MINIMUM-NAME-LENGTH FROM 6 TO 4
MAXIMUM-NAME-LENGTH FROM 12 TO 24

This command relaxes previously specified lengths of primary names of elements.

2. ALTER-META-ENTITY-TYPE
MEDIUM
ALTERNATE-ENTITY-TYPE-NAME FROM *null TO MEDIA

This command allows the entity-type named MEDIUM to be addressed also by the name MEDIA.

3. ALTER-META-ENTITY-TYPE
MULTIPLEXOR-USES-CIRCUIT
INVERSE NAME FROM CIRCUIT-USED-BY-MULTIPLEXOR
TO CIRCUIT-ASSIGNED-TO-MULTIPLEXOR

This command changes the inverse name of the relationship-type MULTIPLEXOR-USES-CIRCUIT.

4. ALTER-META-ENTITY-TYPE
FILE-ACCESS-METHODS
DATA-VALUE FROM NEW-METHOD TO *null
DATA-VALUE FROM *null TO NEW-METHOD-1
DATA-VALUE FROM *null TO NEW-METHOD-2

This command modifies the values of the meta-entity FILE-ACCESS-METHODS which is an instance of the meta-entity-type ATTRIBUTE-TYPE-VALIDATION-DATA, and which has as meta-attributes the valid values that can be used for the attribute-type ACCESS-METHOD for entities of the type FILE. The result of the command is to delete the value NEW-METHOD, and to add the values NEW-METHOD-1 and NEW-METHOD-2.

5.1.5 ALTER-META-ENTITY-WITH-LOCK COMMAND

PURPOSE:

- (a) To alter a meta-entity for which the meta-attribute-type INSTALLATION-LOCK has the value ON, and
- (b) for a specified meta-entity, to alter the value assigned to the meta-attribute-type INSTALLATION-LOCK.

The specification of this command is identical to that of the ALTER-META-ENTITY-TYPE command with the following changes:

- (a) Rule 5. does not apply.
- (b) INSTALLATION-LOCK is deleted from the list in Rule 6.
- (c) Error Condition 5. does not apply.

5.1.6 ALTER-META-RELATIONSHIP COMMAND

PURPOSE: To modify meta-attributes of an existing meta-relationship.

FORMAT: ALTER META-RELATIONSHIP
meta-entity-name-1 TO meta-entity-name-2
clause-1 [,clause-2] ...

RULES:

1. The specified meta-relationship must exist in the schema.
2. The meta-attribute clauses are of the form

meta-attribute-type FROM value-1 TO value-2

to indicate the modification that is to take place. A meta-attribute is deleted by specifying

FROM value-1 TO *null

and a meta-attribute is added by specifying

FROM *null TO value-2

Whenever value-1 is not null, this value must exist in the schema.

ACTIONS PERFORMED:

1. The specified meta-attributes are modified.
2. The command is recorded in the log/audit file.
3. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. There does not exist a meta-entity with name meta-entity-name-1 in the schema.
2. There does not exist a meta-entity with name meta-entity-name-2 in the schema.
3. The specified meta-relationship does not exist in the schema.

EXAMPLE

ALTER-META-RELATIONSHIP
PROGRAM TO PROGRAMMER-NAME
SINGULAR/PLURAL FROM SINGULAR TO PLURAL

It is assumed here that PROGRAMMER-NAME is the name of an attribute-type and this command specifies that multiple instances of this attribute-type are to be

allowed for any one instance of a PROGRAM entity.

5.1.7 CHANGE-META-NAME COMMAND

PURPOSE: To change the name of an existing meta-entity.

FORMAT: CHANGE-META-NAME
meta-entity-name-1 TO meta-entity-name-2

RULES:

1. meta-entity-name-1 is the name of a meta-entity in the schema.
2. There does not exist in the schema a meta-entity with name meta-entity-name-2.

ACTIONS PERFORMED:

1. A meta-entity with name meta-entity-name-2 is created in the schema. This meta-entity has the same meta-attributes and participates in the same meta-relationships as the meta-entity with name meta-entity-name-1.
2. The meta-entity with name meta-entity-name-1 is deleted from the schema.
3. The command is recorded in the log/audit file.
4. Notification of the completion of the execution of the command is given to the user.

EXAMPLE

CHANGE-META-NAME
SYSTEM-ELEMENT TO SYSTEM-COMPONENT

Assuming that SYSTEM-ELEMENT is the name of an entity-type, this command changes the name of this entity-type to SYSTEM-COMPONENT.

5.1.8 CREATE-META-ENTITY COMMAND

PURPOSE: To create a meta-entity and a set of associated meta-attributes in the schema.

FORMAT: CREATE-META-ENTITY
meta-entity-type meta-entity-name
[clause-1] [,clause-2]

RULES:

1. meta-entity-type must be one of the following:

ENTITY-TYPE
RELATIONSHIP-TYPE
RELATIONSHIP-CLASS-TYPE
ATTRIBUTE-TYPE
ATTRIBUTE-GROUP-TYPE
ATTRIBUTE-TYPE-VALIDATION-DATA
STATUS-NAME
STAGE-NAME

2. meta-entity-name must not exist in the schema as the name of a meta-entity.
3. clause-1, ...are meta-attribute clauses and are of the form:

meta-attribute-type = meta-attribute

4. The following meta-attribute-types may not appear in a meta-attribute clause:

DATE-CREATED-IN-SCHEMA

CREATED-IN-SCHEMA-BY
DATE-LAST-MODIFIED-IN-SCHEMA
LAST-MODIFIED-IN-SCHEMA-BY
NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA
BASIC/EXTENDED
SYSTEM-LOCK
STATUS-LOCK

5. If meta-entity-type is ENTITY-TYPE, the following meta-attribute-types may appear in a meta-attribute clause:

INSTALLATION-LOCK
PURPOSE
MINIMUM-NAME-LENGTH
MAXIMUM-NAME-LENGTH
PICTURE
SYSTEM-GENERATED
CONNECTABLE
ALTERNATE-ENTITY-TYPE-NAME
ENTITY-CLASS

6. If meta-entity-type is RELATIONSHIP-TYPE the following meta-attribute-types may appear in a meta-attribute clause:

INSTALLATION-LOCK
PURPOSE
INVERSE-NAME
SEQUENCED
SEQUENCE-PARAMETER

7. If meta-entity-type is RELATIONSHIP-TYPE the following meta-attribute-type cannot appear in a meta-attribute clause:

STATUS-RELATED

8. If meta-entity-type is RELATIONSHIP-CLASS-TYPE the following meta-attribute-types may appear in a meta-attribute clause:

INSTALLATION-LOCK
PURPOSE

9. If meta-entity-type is ATTRIBUTE-TYPE the following meta-attribute-types may appear in a meta-attribute clause:

INSTALLATION-LOCK
PURPOSE
MINIMUM-LENGTH
MAXIMUM-LENGTH
PICTURE
SYSTEM-GENERATED
ALTERNATE-ATTRIBUTE-TYPE-NAME

10. If meta-entity-type is ATTRIBUTE-GROUP-TYPE the following meta-attribute-types may appear in a meta-attribute clause:

INSTALLATION-LOCK
PURPOSE

11. If meta-entity-type is ATTRIBUTE-TYPE-VALIDATION-DATA the following meta-attribute-type must appear in a meta-attribute clause:

VALUE/RANGE

12. If meta-entity-type is ATTRIBUTE-TYPE-VALIDATION-DATA the following meta-attribute-types may appear in a meta-attribute clause:

PURPOSE
DATA-VALUE
DATA-RANGE

where either a meta-attribute clause for DATA-VALUE or DATA-RANGE may appear, but not both.

13. If meta-entity-type is STATUS-NAME the following meta-attribute-types may appear in a meta-attribute clause:

PURPOSE
DEFAULT-STATUS

14. If meta-entity-type is STAGE-NAME the following meta-attribute-type may appear in a meta-attribute clause:

PURPOSE

15. No parametric clauses exist for this command.

ACTIONS PERFORMED:

1. The meta-entity is created in the schema.
2. The following meta-attributes for this meta-entity are created in the schema:

DATE-CREATED-IN-SCHEMA = (time and date of
creation)

CREATED-IN-SCHEMA-BY = (identification of
person issuing
command)

DATE-LAST-MODIFIED-IN-SCHEMA = *null

LAST-MODIFIED-IN-SCHEMA-BY = *null

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = 0

BASIC/EXTENDED = EXTENDED

SYSTEM-LOCK = OFF

STATUS-LOCK = NO

CONTROLLED/UNCONTROLLED = UNCONTROLLED

3. The meta-attributes specified in the meta-attribute clauses are created in the schema.
4. Meta-relationships are created between the given meta-entity and the following meta-entities representing attribute-types:

DATE-CREATED

CREATED-BY

LAST-MODIFICATION-DATE

LAST-MODIFIED-BY

NUMBER-OF-MODIFICATIONS

DESCRIPTION

COMMENTS

5. If the given meta-entity represents an entity-type, additional meta-relationships are created between the given meta-entity and the following meta-entities representing an attribute-type and attribute-group-type, respectively:

CLASSIFICATION
IDENTIFICATION-NAMES

6. If the given meta-entity represents an entity-type, a relationship-type, whose first member is the entity-type ACCESS-CONTROLLER and whose second member is the entity-type being created, will be established. The name of this relationship-type is ACCESS-CONTROLLER-FOR-<meta-entity-name>.
7. If the given meta-entity represents a STAGE-NAME, the meta-entity-name specified also becomes an attribute of attribute-type STAGE.
8. The command is recorded in the log/audit file.
9. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. meta-entity-type is not as per Rule 1.
2. meta-entity-name exists in the schema as the name of a meta-entity.
3. A meta-attribute-clause is specified that contains one of the following meta-attribute-types:

DATE-CREATED-IN-SCHEMA
CREATED-IN-SCHEMA-BY
DATE-LAST-MODIFIED-IN-SCHEMA
LAST-MODIFIED-IN-SCHEMA-BY
NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA
BASIC/EXTENDED
SYSTEM-LOCK

STATUS-LOCK
CONTROLLED/UNCONTROLLED

4. A meta-attribute clause exists which does not apply to the meta-entity-type specified.
5. The meta-entity-type RELATIONSHIP-TYPE is specified and a meta-attribute clause involving the meta-attribute-type STATUS-RELATED exists.
6. The meta-entity-type ATTRIBUTE-TYPE-VALIDATION-DATA is specified and no meta-attribute clause for the meta-attribute-type VALUE/RANGE is given.
7. The meta-entity-type ATTRIBUTE-TYPE-VALIDATION-DATA is specified and there exist meta-attribute clauses for both the meta-attribute-types DATA-VALUE and DATA-RANGE.
8. A parametric clause is specified.

EXAMPLES

1. CREATE-META-ENTITY
ATTRIBUTE-TYPE TRANSACTION-TYPE
PURPOSE = "TO DENOTE THE TYPE OF A TRANSACTION.",
MINIMUM-LENGTH = 2,
MAXIMUM-LENGTH = 3

This command creates an attribute-type named TRANSACTION-TYPE in the schema, and specifies that attributes of this type shall be at least 2 characters in length, but shall not exceed 3 characters in length.

2. CREATE-META-ENTITY
ATTRIBUTE-TYPE-VALIDATION-DATA REPORT-TYPE-VALUES
VALUE/RANGE = VALUE,
DATA-VALUE = 01 "EXTERNAL",
DATA-VALUE = 02 "INTERNAL"

This command creates an attribute-type-validation-data meta-entity named REPORT-TYPE-VALUES in the schema, specifies that this meta-entity will be used for validation of data values (as opposed to ranges) and further specifies that there are to exist two valid values, namely 01 and 02, with transliterated values EXTERNAL and INTERNAL, respectively.

5.1.9 CREATE-META-RELATIONSHIP COMMAND

PURPOSE: To create meta-relationships and associated meta-attributes in the schema.

FORMAT: CREATE-META-RELATIONSHIP
meta-entity-name-1 AND 'meta-entity-name-2
[clause-1] [,clause-2] ...

RULES:

1. meta-entity-1 and meta-entity-2 must exist in the schema as names of meta-entities.
2. The meta-relationship to be created does not already exist in the schema.
3. The meta-relationship defined by meta-entity-name-1 and meta-entity-name-2 must be an instance of a meta-relationship-type listed in Section 3.1.2 of this specification.
4. clause-1 ... is of the form

meta-attribute-type-1 = meta-attribute-1
5. meta-attribute-types given must apply to the meta-relationship being created, as specified in Section 3.1.4.
6. If one of the meta-entities specified is the attribute-type-validation-procedure VALUE-VALIDATION, the other meta-entity must be of type attribute-type. In that case it is required that there exist a meta-relationship with this attribute-type as one member and an attribute-type-validation-data with VALUE/RANGE = VALUE as the other member.
7. If one of the meta-entities specified is the attribute-type-validation-procedure RANGE-VALIDATION, the other meta-entity must be of type attribute-type.

In that case it is required that there exist a meta-relationship with this attribute-type as one member and an attribute-type-validation-data with VALUE/RANGE = RANGE as the other member.

8. If the meta-relationship is of the type

M-R-T(RELATIONSHIP-TYPE, ENTITY-TYPE)

a meta-attribute for the meta-attribute-type POSITION must be specified. The value assigned cannot already exist in the schema for the named relationship-type.

9. If the meta-relationship is of the type

M-R-T(ATTRIBUTE-GROUP-TYPE, ATTRIBUTE-TYPE)

a meta-attribute for the meta-attribute-type GROUP-POSITION must be specified.

ACTIONS PERFORMED:

1. The meta-relationship and associated meta-attributes are created in the schema.
2. The command is recorded in the log/audit file.
3. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. A meta-entity with name meta-entity-name-1 does not exist in the schema.
2. A meta-entity with name meta-entity-name-2 does not exist in the schema.
3. The meta-relationship specified is not an instance of a meta-relationship-type which exists in the schema model.

4. A meta-attribute clause is not in the required form.
5. A meta-attribute-type is specified which does not apply to the given meta-relationship-type.
6. An erroneous specification of the meta-attribute for SINGULAR/PLURAL is given.
7. The required attribute-type-validation-data meta-entity does not exist in the schema (Rules 7. and 8.).
8. A parametric clause is specified.

EXAMPLES

1. CREATE-META-RELATIONSHIP
FILE AND OBSOLESCENCE-DATE
SINGULAR/PLURAL = SINGULAR

This command specifies that entities of type FILE have attributes of type OBSOLESCENCE-DATE, and furthermore that any such entity can have at most one such attribute.

2. CREATE-META-RELATIONSHIP
LOCATION-HAS-DOCUMENTATION AND LOCATION
POSITION = FIRST

CREATE-META-RELATIONSHIP
LOCATION-HAS-DOCUMENTATION AND DOCUMENTATION
POSITION = SECOND

These two commands specified that the meta-entity LOCATION-HAS-DOCUMENTATION of type relationship-type has the entity-type LOCATION as the first member, and the entity-type DOCUMENTATION as the second member.

5.1.10 REPLACE-META-RELATIONSHIP COMMAND

PURPOSE: To replace one meta-relationship by another. This command, which is a combination of ABOLISH-META-RELATIONSHIP and CREATE-META-RELATIONSHIP commands, is required to replace one attribute-type-validation-data by another. Carrying out this procedure in two separate steps would violate the rule that there cannot exist in the schema a meta-entity of type attribute-type which was a member of a meta-relationship whose other member was an attribute-type-validation-procedure meta-entity without also being a member of a meta-relationship whose other member is an attribute-type-validation-data meta-entity.

FORMAT: REPLACE-META-RELATIONSHIP
meta-entity-name-1 FROM meta-entity-name-2
TO meta-entity-name-3
[NO-META-ATTRIBUTES]

RULES:

1. meta-entity-name-1, meta-entity-name-2, and meta-entity-name-3, are names of meta-entities in the schema.
2. There exists a meta-relationship in the schema with members meta-entity-name-1 and meta-entity-name-2.
3. There does not exist a meta-relationship in the schema with members meta-entity-name-1 and meta-entity-name-3.
4. The meta-relationship-type of meta-entity-name-2 and meta-entity-name-3 must be the same.
5. Since this command specifies the deletion of the meta-relationship with members meta-entity-name-1 and meta-entity-name-2, this meta-relationship may not be deleted if the deletion affects the information contents of the dictionary.

6. If the meta-entity-type of meta-entity-name-2 and meta-entity-name-3 is attribute-type-validation-data (and consequently the meta-entity-type of meta-entity-name-1 is attribute-type) the command cannot violate the integrity of existing instances of this attribute-type.
7. If meta-entity-name-2 is an attribute-type-validation-data meta-entity with DATA/VALUES = DATA, then meta-entity-name-2 (which must have the same type) must have the same meta-attribute.
8. If meta-entity-name-2 is an attribute-type-validation-data meta-entity with DATA/VALUES = RANGE then meta-entity-name-2 (which must have the same type) must have the same meta-attribute.

ACTIONS PERFORMED:

1. The meta-relationship with members meta-entity-name-1 and meta-entity-name-3 is created and the meta-relationship with members meta-entity-name-1 and meta-entity-name-2 is deleted.
2. If [NO-META-ATTRIBUTES] is not specified, the new meta-relationship will have the same meta-attributes as the one being replaced.
3. The command is recorded in the log/audit file.
4. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. There does not exist a meta-entity with name meta-entity-name-1 in the schema.
2. There does not exist a meta-entity with name meta-entity-name-2 in the schema.

3. There does not exist a meta-entity with name meta-entity-name-3 in the schema.
4. There does not exist in the schema a meta-relationship with members meta-entity-name-1 and meta-entity-name-2.
5. There does exist in the schema a meta-relationship with members meta-entity-name-1 and meta-entity-name-3.
6. The meta-entity-type of meta-entity-name-2 and the meta-entity-type of meta-entity-name-3 are not the same.
7. Attribute-type-validation-data for an attribute-type is being specified which does not agree with attributes of that attribute-type existing in the dictionary.
8. A meta-attribute clause is specified.
9. A parametric clause is specified.

EXAMPLE

1. REPLACE-META-RELATIONSHIP
TRANSACTION-CODE FROM TRANSACTION-CODE-VALUES
TO NEW-TRANSACTION-CODE-VALUES

This command is intended to replace the attribute-type-validation-data meta-entity TRANSACTION-CODE-VALUES by the meta-entity NEW-TRANSACTION-CODE-VALUES in the meta-relationship whose other member is the attribute-type TRANSACTION-CODE. The command will not be allowed to execute whenever there exists in the dictionary an attribute of TRANSACTION-CODE which exists in TRANSACTION-CODE-VALUES but does not exist in NEW-TRANSACTION-CODE-VALUES.

5.2 SCHEMA REPORTING COMMANDS

The following commands are available for reporting on the contents of the schema. The order of presentation of the commands is alphabetic by command name.

5.2.1 IS-META-RELATED COMMAND

PURPOSE: To inform a user whether or not two meta-entities are "directly related" or "indirectly related".

FORMAT: IS-META-RELATED
meta-entity-name-1 AND meta-entity-name-2

RULES:

1. meta-entity-name-1 and meta-entity-name-2 must be the names of two distinct meta-entities in the schema.

ACTIONS PERFORMED:

1. If the two meta-entities specified are "directly related", the message

"YES, DIRECTLY"

is returned to the user.

2. If the two meta-entities specified are "indirectly related", the message

"YES, INDIRECTLY"

is returned to the user.

3. If the two meta-entities specified are not related, the message

"NEITHER DIRECTLY NOR INDIRECTLY"

is returned to the user.

ERROR CONDITIONS:

1. meta-entity-name-1 and meta-entity-name-2 are not the names of two distinct meta-entities in the schema.
2. An invalid meta-attribute clause is specified.
3. An invalid parametric clause is specified.

EXAMPLES

1. IS-META-RELATED
ACCESS-METHOD AND ACCESS-METHOD-VALUES

The response would be to show that the attribute-type ACCESS-METHOD and the attribute-type-validation-data meta-entity ACCESS-METHOD-VALUES are directly related.

5.2.2 META-CATALOG COMMAND

PURPOSE: To produce a report on the contents of the entire schema.

FORMAT: META-CATALOG
[Destination]

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. A report is produced, the contents of which is the entire schema. The format of the report is similar to the format of the META-LIST command.
2. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. An meta-attribute clause is specified.
2. A parametric clause is specified.

5.2.3 META-LIST COMMAND

PURPOSE: To produce a report on the schema descriptors associated with a set of meta-entities.

FORMAT: META-LIST
Report Set
[Destination]

where Report Set is one of the following:

- a) meta-entity-type-name
- b) meta-entity-name-1 [,meta-entity-name-2] ...

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. meta-entity-type-name must be one of the names given in 3.1.1.
2. meta-entity-name-1 ... must be the name(s) of a meta-entity in the schema.
3. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. A report is produced, in which, for every meta-entity in the report set, the following is shown:
 - a) The name of the meta-entity and its type,
 - b) the meta-attributes of the meta-entity, and
 - c) each meta-relationship in which the meta-entity is a member together with its meta-attributes and the name of the other meta-entity which is a member of that meta-relationship.
2. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. meta-entity-type-name is not a valid name.

2. A name is specified which is not the name of a meta-entity in the schema.
3. A meta-attribute clause is specified.
4. A parametric clause is specified.

EXAMPLES

1. META-LIST
ACCESS-METHOD-VALUES

This command will produce a report showing all the valid attributes of the attribute-type ACCESS-METHOD which exist as meta-attributes of the attribute-type-validation-data meta-entity ACCESS-METHOD-VALUES. If there exist both codes and transliterated values, both will be shown, as for example the following:

```
ATTRIBUTE-TYPE-VALIDATION-DATA
ACCESS-METHOD-VALUES
01    "RANDOM"
02    "SEQUENTIAL"
03    "METHOD-A"
```

Additionally, the report will show that there exists a meta-relation with members: the attribute-type ACCESS-METHOD and the meta-entity being reported on.

2. META-LIST
FILE

This command produces a report on the entity-type FILE, its meta-attributes and the meta-relationships in which it is a member.

5.2.4 META-TRACE COMMAND

PURPOSE: To produce a report, for a specified meta-entity, on all meta-entities which are:

- a) "directly related", i.e., are members of a meta-relationship where the specified meta-entity is the other member, or
- b) "indirectly related", i.e., where there exists a sequence of meta-relationships, MR_1, \dots, MR_n , such that the specified meta-entity is a member of MR_1 , and for each k , there is a meta-entity which is a member of MR_k and $MR_{(k+1)}$.

FORMAT: META-TRACE
meta-entity-name
[Destination]

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. meta-entity-name must be the name of a meta-entity in the schema.
2. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. A report is produced on a set of meta-entities which can be described as the combined output of multiple applications of the META-LIST command.

- a) The report for the specified meta-entity.
- b) The report for each previously unreported meta-entity.
- c) Step b) is repeated as long as previously unreported meta-relationships exist.

The order in which the meta-entities are listed is to follow each sequence of meta-relationships to its end.

2. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. meta-entity-name is not the name of a meta-entity in the schema.
2. A meta-attribute clause is specified.
3. A parametric clause is specified.

CHAPTER 6. COMMANDS FOR INTERACTION WITH THE DICTIONARY

This chapter contains the specification of the commands that are available for interaction with the dictionary. The chapter is organized into five major sections:

1. General rules that apply to these commands.
2. Qualification commands, which can be used to build lists of entities that can be used by reporting, query, and some maintenance commands.
3. The commands required to create entities, relationships, and attributes in the dictionary, and to maintain them.
4. The commands that are available for producing reports from the dictionary.
5. Query commands that are available for interrogation of the dictionary.

All commands specified in this chapter are subject to the security provisions specified in Chapter 8, and the rules for defaults specified in Chapter 2 are in effect.

6.1 GENERAL RULES

This section contains general rules that are used throughout this chapter.

6.1.1 ENTITY-NAME RULES

The dictionary schema contains optional clauses that allow installation standards to be enforced on the primary names that are assigned to entities of a given entity-type.

For each entity-type:

1. The meta-attribute clause

MINIMUM-NAME-LENGTH = integer-1

specifies that the primary name of an entity of this entity-type must have at least integer-1 characters.

2. The meta-attribute clause

MAXIMUM-NAME-LENGTH = integer-2

specifies that the primary name of an entity of this entity-type can have at most integer-2 characters.

3. The meta-attribute clause

PICTURE = form-1 [,form-2] ...

where form-i is a string of characters A, N, and X (and possibly other implementor-defined characters) that indicate whether the character of a primary name in a given position must be

alphabetic	(A)
numeric	(N)
alphanumeric	(X)

Multiple forms may be specified, in which case a name specified must be in one of the forms.

For example, the clause

PICTURE = AANNXX

would specify that XY1234 and AB987X are valid primary

names for entities of the given entity-type, but that XYZ142 and A1247Q are not valid names.

The specifications given in various clauses must be consistent. It may be expected that an installation would not use both the length and picture clauses for any one entity-type. Two sets of such specifications would be redundant, and could conceivably be self-contradictory, as would be the case if, for example, it were specified that

MINIMUM-NAME-LENGTH = 3
MAXIMUM-NAME-LENGTH = 3

as well as

PICTURE = AAAA

This specification would cause every name to be rejected as it is impossible for a name to be exactly 3 characters long and to have the specified picture.

Primary names of entities specified in the ADD-ENTITY, ADD-RELATIONSHIP, COPY, DECLARE, and RENAME commands are subject to rules expressed with these clauses.

6.1.2 USE-AS-IDENTIFIER ATTRIBUTE-TYPE RULES

The system-standard schema contains the attribute-type SHORT-NAME for entities of type ELEMENT. The meta-relationship that exists in the system-standard schema with members ELEMENT and SHORT-NAME has the value YES for the meta-attribute-type USE-AS-IDENTIFIER, denoting that the system will insure that attributes of type SHORT-NAME will be unique in the dictionary. This means that for an entity of type ELEMENT, not only is the primary name of the entity unique in the dictionary, but also the attribute SHORT-NAME, should such attribute have been assigned.

In the following maintenance commands:

DELETE-ENTITY
DELETE-RELATIONSHIP

MODIFY-ENTITY
MODIFY-RELATIONSHIP
CHANGE-STATUS
RENUMBER

the general rule exists that the value of the attribute-type SHORT-NAME can be used in place of the primary name of an entity of type ELEMENT. The same rules for version numbers apply to this value as to the primary name of an entity. Whenever this value is used, it is to be prefixed with the character \$, this character being an implementor defined symbol.

For sake of simplicity, this value of the attribute-type SHORT-NAME of an element will be referred to as the **short-name** of the element.

For example, if there is an element with primary name NUMBER-OF-DEPENDENTS and short-name NO-DEP, a command to modify this entity instead of being

MODIFY-ENTITY
NUMBER-OF-DEPENDENTS

followed by the clauses specifying the modifications, could be stated as

MODIFY-ENTITY
\$NO-DEP

followed by these same clauses.

Short-names can be used in REPORT and QUERY commands through the use of the QUALIFY command. For example,

QUALIFY SHORT-NAME = NO-DEP
SHORT-NAME = NO-DEP-M
SHORT-NAME = NO-DEP-F

builds a qualification list whose members are the elements with short-name NO-DEP, NO-DEP-M, NO-DEP-F. A detailed description of this command is given in Section 6.2.

The facilities discussed here can be extended using the extensibility facilities that have been specified in Chapter 3.

Two primary options exist:

1. A meta-relationship can be specified, one member of which is an entity-type (e.g., SYSTEM), with the other member being the attribute-type SHORT-NAME, and the value of the meta-attribute-type USE-AS-IDENTIFIER of this meta-relationship can be specified as YES. This will cause the use of short-names to be extended to entities of this entity-type in the identical manner as it exists for the entity-type ELEMENT.
2. An attribute-type, for example STANDARD-IDENTIFIER, can be defined in the schema, and meta-relationships can be set up as in 1. above. This would allow the values of STANDARD-IDENTIFIER to be used in the same manner as the short-name. Any such attribute-type will be referred to as an **use-as-identifier attribute-type**.

6.1.3 ATTRIBUTE RULES

The dictionary schema contains:

1. Optional clauses that allow an installation to specify the minimum number of characters that an attribute of a given attribute-type must have.
2. Optional clauses that allow an installation to specify the maximum number of characters that an attribute of a given attribute-type can have.
3. Optional clauses that allow an installation to specify the PICTURE of attributes of a given attribute-type.
4. Optional clauses that allow an installation to control the number of times that attributes of a given attribute-type can be assigned to an entity or relationship of a specified type.
5. Optional clauses that allow an installation to control the number of times that attribute-groups of a given

attribute-group-type can be assigned to an entity or relationship of a specified type.

6. Facilities that allow attributes of a given attribute-type to be restricted to a set of pre-defined values.
7. Facilities that allow attributes of a given attribute-type to be restricted to a set of pre-defined ranges of values.

Specifically, these features may be invoked through the following:

1. The meta-attribute clause

MINIMUM-LENGTH = integer-1

specifies that an attribute of a given attribute-type must contain at least integer-1 characters.

2. The meta-attribute clause

MAXIMUM-LENGTH = integer-2

specifies that an attribute of a given attribute-type can contain at most integer-2 characters.

3. The meta-attribute clause

PICTURE = {TEXT|form-1[,form-2]...}

where form-i is as in 6.1.1 (entity-name rules), and allows the specification of the kinds of characters that make up an attribute of a given attribute-type. Rules for TEXT attributes are given in 6.1.4. Whenever TEXT is not specified, multiple forms may be given, in which case an attribute specified must be in one of the forms.

4. The meta-relationships that assign an attribute-type to an entity-type or a relationship-type have meta-attribute-types SINGULAR/PLURAL and MAXIMUM-NUMBER-OF-OCCURRENCES that can be used to control the number of occurrences of an attribute of a given attribute-type

for an entity of a given entity-type or a relationship of a given relationship.

For example, let TYPE-A denote an entity-type, and let NAME-X be an attribute-type that is assigned to TYPE-A in the schema, and assume that for the meta-relationship that expresses this assignment the clause

SINGULAR/PLURAL = SINGULAR

has been given. This means that for every instance of TYPE-A there can be at most one attribute of NAME-X. If instead the clauses

SINGULAR/PLURAL = PLURAL

MAXIMUM-NUMBER-OF-OCCURRENCES = 5

had been given, this would denote that 5 such attributes were permissible. This facility works in an identical manner with relationships.

It should be noted that the number of occurrences for an attribute being specified is specific to an entity-type or relationship-type. For example, suppose that the attribute-type used above is not only assigned to the entity-type TYPE-A, but also to the entity-type TYPE-Q. It would then be possible to specify a maximum of 5 occurrences for TYPE-A and a maximum of only 3 occurrences for TYPE-Q, or any combination of the meta-attributes in the clauses.

5. Clauses identical to the ones discussed above exist to control the number of occurrences of attribute-groups in entities and relationships of a given type.
6. The schema contains an attribute-type-validation-procedure called DATA-VALUES, as well as facilities for establishing schema descriptors for attribute-type-validation-data. Whenever both the DATA-VALUES procedure and such a descriptor are assigned to an attribute-type, all attributes of this type are required to be present in the attribute-type-validation-data. The maximum number of such values that can be stored in the schema for any one attribute-type-

validation-data meta-entity is implementor defined.

7. Similar facilities to the one above exist to restrict attributes of a given attribute-type to be in one of a set of pre-defined ranges. In this case there is an attribute-type-validation-procedure called RANGE, and the corresponding attribute-type-validation-data descriptor contains the ranges within which attributes must fall.

Attribute-clauses and attribute-group clauses in the ADD-ENTITY, ADD-RELATIONSHIP, DECLARE, MODIFY-ENTITY, and MODIFY-RELATIONSHIP commands are subject to the rules given.

6.1.4 TEXT ATTRIBUTE-TYPES

It can be specified that the meta-attribute-type PICTURE of an attribute-type has the value TEXT. These attributes are created and maintained in a manner different from other attributes.

Each attribute of such an attribute-type consists of a set of lines, each line having a line number. The set of lines is ordered in an ascending order by line number. The number of characters in a line is implementor-defined, as well as the maximum number of lines that can exist in a single attribute.

An attribute of this kind is specified in an ADD-ENTITY or ADD-RELATIONSHIP command by a clause in the following form:

```
attribute-type-name
[START = integer-1] [INCREMENT = integer-2]
<text-string of line>
[<text-string of line>]
.
.
.
```

Each line entered is assigned a system-generated line number according to the following rules:

1. If the optional START subclause, where integer-1 must

be a positive integer, is given, the first line is assigned the line number integer-1. The default case results in the assignment of an implementor-specified positive integer.

2. If the optional INCREMENT subclause, where integer-2 must be a positive integer, is given, successive line numbers are formed by incrementing the previous line number by integer-2. In the default case, line numbers are incremented by an implementor-specified positive integer.

An attribute of this kind can be changed by a MODIFY-ENTITY or MODIFY-RELATIONSHIP command with a clause in the following form:

```
attribute-type-name  
[DELETE integer-1 [THROUGH integer-2]] ...  
[integer-3 <text-string>] ...
```

The following RULES apply:

1. Specification of the optional subclause DELETE integer-1 without the THROUGH subclause will cause the line with line number integer-1 to be deleted.
2. If both integer-1 and integer-2 are specified, then integer-2 must be greater than integer-1.
3. Specification of this subclause with the THROUGH subclause will cause deletion of all existing lines with line numbers in the range integer-1 to integer-2, including the lines with line numbers integer-1 and integer-2, if they exist.
4. The optional subclause integer-3 <text-string> has two possible effects:
 - a) If there exists a line with line number integer-3, the text string specified will replace the text string in existence on that line.
 - b) If there does not exist a line with line number integer-3, the line specified is inser-

ted in the numerical sequence of the existing lines.

5. At least one subclass must be specified in a clause, and any desired number of each form may be given. All DELETE subclasses must precede the other subclasses.

Existing line numbers may be changed by the RENUMBER command.

6.1.5 RULES FOR USAGE-NAMES ATTRIBUTE-GROUPS

As discussed in Chapter 4, the relationship-class-type CONTAINS in the system-standard schema is composed of the following relationship-types:

FILE-CONTAINS-FILE
FILE-CONTAINS-RECORD
FILE-CONTAINS-ELEMENT
RECORD-CONTAINS-ELEMENT
RECORD-CONTAINS-RECORD
ELEMENT-CONTAINS-ELEMENT
DOCUMENT-CONTAINS-DOCUMENT
DOCUMENT-CONTAINS-RECORD
DOCUMENT-CONTAINS-ELEMENT
SYSTEM-CONTAINS-SYSTEM
SYSTEM-CONTAINS-PROGRAM
SYSTEM-CONTAINS-MODULE
PROGRAM-CONTAINS-PROGRAM
PROGRAM-CONTAINS-MODULE
MODULE-CONTAINS-MODULE

Each one of these relationship-types is associated with the attribute-group-type USAGE-NAMES, which is composed of attribute-types with name alternate-name-1 and alternate-name-2. Special rules exist for this attribute-group-type, as follows:

Let R-T denote one of the relationship-types in the preceding list, and let T-1 and T-2 denote the entity-types involved in the relationship. The attribute-types alternate-name-1 and alternate-name-2, in this case, correspond to alternate names of entities of entity-types T-1 and T-2, respectively.

Let entity-A and entity-B denote entities of entity-types T-1 and T-2, respectively, and suppose that the relationship of type R-T has entity-A and entity-B as members. Further assume that this relationship has an instance of the attribute-group-type USAGE-NAMES which is composed of the values name-A and name-B. The meaning that is attached to this attribute-group is as follows:

In the relationship where entity-A contains entity-B, when entity-A has the alternate name name-A, then entity-B has the alternate name name-B.

In the context discussed here, a primary name of an entity can be used in place of an alternate name of that entity in this attribute-group. Another way of stating this is that the primary name of an entity automatically qualifies as an alternate name.

The attribute-group-type USAGE-NAMES is used only for the relationship-class-type CONTAINS in the system-standard schema; however, extensibility facilities can be used to extend its usage to other relationship-types.

The preceding can be illustrated by the following example.

Suppose that there is an instance of the RECORD entity-type called PAYROLL-RECORD which CONTAINS an ELEMENT called PAY-NUMBER. In the data processing environment, however, the PAYROLL-RECORD is known as PAY-REC and PAY-NUMBER is known as PAY-NO. In this case the relationship PAYROLL-RECORD CONTAINS PAY-NUMBER has the attribute-group (PAY-REC,PAY-NO), denoting that when PAYROLL-RECORD is known as PAY-REC, PAY-NUMBER is known as PAY-NO. Multiple instances of this attribute-group may exist for the same relationship. Suppose that in another file the RECORD PAYROLL-RECORD is used with the name R1234, and the ELEMENT PAY-NUMBER is used with the name EL456P; this can be expressed by the attribute-group (R1234,EL456P) in the relationship between the RECORD and ELEMENT instances.

Equally, in another file PAYROLL-RECORD may be used with name P-R and PAY-NUMBER may be known by its primary name. Then the attribute-group of type USAGE-NAMES would be (P-R,PAY-NUMBER), where it would not be necessary to declare PAY-NUMBER as being an alternate name for itself.

The following rules apply to the attribute-group-type USAGE-NAMES:

1. If the attribute-group (name-1,name-2) is entered into the dictionary for a CONTAINS relationship with members entity-1 and entity-2, name-1 is also entered as an attribute of type ALTERNATE-NAME of entity-1, and name-2 is also entered as a like attribute of entity-2.
2. If the above attribute-group is deleted, each of the above named attributes of type ALTERNATE-NAME is also deleted, unless it is a member of another attribute-group of type USAGE-NAMES of the same relationship.
3. If one of the names in the attribute-group is modified, this name is also modified in the ALTERNATE-NAME attribute of the entity used, unless the unmodified name is also used in another attribute-group of type USAGE-NAMES of the same relationship. In this case, the new name is added to the list of alternate names of the entity.
4. If a command to modify or delete an attribute of type ALTERNATE-NAME of an entity is attempted and if the attribute is used in an attribute-group of type USAGE-NAMES for a relationship involving this entity, it will be disallowed.
5. The primary name of an entity can be used in an attribute-group of type USAGE-NAMES in place of an alternate name of that entity.
6. A command to rename an entity whose primary name is used in an attribute-group of type USAGE-NAMES for a relationship involving this entity will be disallowed.

6.1.6 AUDIT-ATTRIBUTE-TYPES

The dictionary schema contains some attribute-types that pertain to all entity-types and relationship-types in the schema, whether they are part of the system-standard schema or they have been created through the use of extensibility facilities. These attribute-types will be referred to as the Audit Attribute-Types:

DATE-CREATED
CREATED-BY
LAST-MODIFICATION-DATE
LAST-MODIFIED-BY
NUMBER-OF-MODIFICATIONS

These attribute-types should not be confused with the meta-attribute-types DATE-CREATED-IN-SCHEMA, CREATED-IN-SCHEMA, etc., which are used to audit actions on the schema.

The audit-attribute-types in the list given above are used to document audit-oriented data about update operations that take place on the dictionary data.

6.1.7 NULL ATTRIBUTES

Null values can be treated in the same manner as any other value of an attribute-type. The system null is represented by *null in this specification, to be replaced by an implementor defined symbol. It can be used in a command, and thus the statement

ATTRIBUTE-TYPE-NAME = *null

will evaluate to true if the value of the attribute-type ATTRIBUTE-TYPE-NAME is the system null. Thus any command that has a qualification component that can be used to test an attribute can substitute *null for that value. It should be noted the *null does not have a type; a test for *null can be made regardless of the defined type of the attribute (alphabetic, numeric, text, etc.).

6.2 QUALIFICATION

During the manipulation of entities in a dictionary, it is often necessary to identify which of the dictionary entities are to be selected. This process of identification is termed qualification, and the result of such an operation is a list of the primary names (or other unique identification) of the relevant entities; this list is termed a qualification list.

Qualification may be achieved either explicitly, by issuing a command that returns a qualification list, or implicitly, by producing the list as a side effect of another command, or incorporating it in the command. Thus the process of reporting may either be of the form:

1. Qualify which entities are to be reported on, giving the qualification list;
2. State the report to be produced, and produce it for all entities on the list.

or else it may be in one part:

1. Make a reporting command that first generates the qualification list internally and then produces the report.

In both cases, the qualification list may be retained, and thus further use may be made of this list in later reporting. Qualification lists may also be used in some maintenance commands.

There are therefore several modes of operation, all of which are equivalent, but which appear different to the user in their achieving the required result:

1. Qualification followed by reporting;
2. Reporting including internal generation of a qualification list but no external manifestation to the user;
3. Reporting including generation of a qualification list

that may be further used by other commands, when named.

The material of this section may therefore be considered to be used in one of three ways:

1. QUALIFY
[optional-qualification-list-name] [clauses]
REPORT-OR-QUERY-COMMAND-using-qualification-list

2. QUALIFY
SAVE name-of-procedure

.
.
.

RUN name-of-procedure
[optional-qualification-list-name]
Report-or-Query-Command-using-qualification-list

3. Report-or-Query-or-Maintenance-Command
[clauses-for-qualification] [AND]
[optional-qualification-list-name] [AND]
[RUN name-of-procedure]

The report commands may use either qualification clauses, a procedure, a qualification list, or a combination of these options.

Normally, when the qualification list is available to the user, it may be changed by further commands. Thus the qualification list A may be modified by a user to produce a new list by:

1. Using list A as an "input" to generate a new qualification list B, which can be a subset of A, or else use list A to produce a list of entirely new entities.
2. Generation of a new qualification list C that is then used to produce another list D as the result of set operations (union, disjoint union, difference, etc.) between A and C.

Either of the above methods will be considered functionally equivalent from the standpoint of the core standard.

However, the discussion in this section assumes that a qualification list is available to the user either by name, or by use of a procedure that generates it, or by indirect reference to the latest version of the qualification list. Thus the alternatives assume seven basic types of commands to operate on or generate a qualification list:

1. Generation of a qualification list (e.g., through use of a QUALIFY command). This command may generate a "current" qualification list, that can be the object of any future command that does not name the required qualification list (this list being referred to as the current-list). If the user wishes, the QUALIFY command may include a name by which the list may later be referenced; if a name is used, the current list will not be changed by the command.
2. Manipulation of two qualification lists using set operations (e.g., UNION, INTERSECTION, and SET-DIFFERENCE). The current-list may participate as one of the sets, and the result, if not named, will become the current list.
3. Deletion of the qualification list (e.g., by using the DISQUALIFY command).
4. Saving of the procedure(s) used to generate the qualification list (e.g., by using the SAVE-LIST command).
5. Executing the procedure to produce a new version of the qualification list (i.e., through the RUN command).
6. Deleting the procedure for generating a qualification list (e.g., by the DROP-PROCEDURE command).
7. Listing all procedures that have been saved.

The QUALIFY command contains eight optional clauses. These may be used in conjunction with, or as substitutes for, the qualification list in a reporting or query command or some maintenance commands; thus each of the eight clauses is discussed separately.

6.2.1 QUALIFY COMMAND

PURPOSE: The qualification of a set of entities depends on the users' need for dictionary data. At its most gross, qualification of the whole dictionary may be needed (e.g., when the entire contents of the dictionary are to be printed according to some format). In the finest grain, only one entity is required. Thus the qualification list may contain from zero to all the primary names of entities in the dictionary.

There are several ways that the user may wish to select and then restrict the qualification, and each of these may be made by using one or more clauses of the qualify command as follows:

1. Selection by entity-type-name (e.g., only FILE and PROGRAM entity-types are to be returned), or by entity-name stated explicitly, or by some combination of each. This will be achieved in a Primary-Name-Selection Clause.
2. Selection by using an alternate-name instead of the primary-name. Because the user may not know that the name is not a primary-name, one form of this clause appears identical to the entity-selection-clause. However, because the user may wish to qualify entities by use of the alternate-name-context, there is a special Alternate-Name-Selection Clause.
3. Restriction of an entity's selection based on some characteristic of its primary name, such as its length, whether it contains, starts or ends with a specific string (character sequence), etc. This is achieved in a Primary-Name-Restriction Clause.
4. Restriction based on entity names and relationship-type and relationship-class-type. Here the entity is rejected because it is not directly connected to another through a named relationship-type, or else because the entity is not indirectly connected to the second entity through a composition of

relationship-types, several applications of the same relationship-type, or a relationship-class-type. Such qualification entails a Relationship-Restriction Clause.

5. Further restriction based on the person who created or last changed the entity or the date on which the entity was created or last changed, or else on the number of changes that have been made. This is achieved in an Audit-Attribute-Restriction Clause.
6. Restriction based on attribute-types. This is provided in an Attribute-Restriction Clause.
7. Restriction of selection based on characteristics (such as the presence of certain character strings) in the attributes of attribute-types composed of TEXT. These are in the optional Text-String-Restriction Clause.
8. Selection through the use of the keyword attribute-type. This is achieved in the Classification-Restriction Clause.

The qualification request thus consists of a series of optional clauses which might typically be provided according to the following format.

FORMAT: QUALIFY
 [qualification-list-name]
 {ALL|
 Primary-Name-Selection Clause|
 Alternate-Name-Selection Clause}
 [Primary-Name-Restriction Clause]
 [Relationship-Restriction Clause]
 [Audit-attribute-Restriction Clause]
 [Attribute-Restriction Clause]
 [Text-String-Restriction Clause]
 [Classification Clause]

RULES:

1. The qualification-list-name is optional. If it is not given, the qualified list becomes the current-list. If it is given, the qualification list is identified by this name and there is no change to the current-list. A current-list can be referenced implicitly (though it has an internal name, it cannot be externally referenced) -- thus any command that needs a qualification list but does not name one is assumed to operate on the current-list.
2. The ALL option allows qualification of every dictionary entity. Thus QUALIFY ALL would produce a qualification list containing every entity existing in the dictionary.
3. The number of entities in the qualification list is returned to the user at the termination of the command.
4. The separate clauses above are considered to be logically connected through AND.
5. All errors are discussed in the specifications of the clauses. However any error will cause abortion of the command, and the current qualification list will be unchanged.

EXAMPLES

1. QUALIFY ALL

This makes the current list a list of all the entities in the dictionary.

2. QUALIFY ZIP ALL

This generates a new list of all the entities in the dictionary and stores it as "ZIP".

6.2.1.1 ENTITY-SELECTION CLAUSE

PURPOSE: Entities are selected either by stating their primary name, or by giving the overall entity-type of interest plus naming any exclusions. Because extensibility may have been used to add new entity-types, the command allows these to be accessed. Because the addition of a new relationship in the dictionary may cause an implicit entity to be added, facilities are required by means of which such implicit entities can be selected or excluded.

FORMAT: {{{EXPLICIT|IMPLICIT}|BOTH}}

[ENTITY-TYPE entity-type-name [EXCLUDE entity-name]...
[,entity-type-name [EXCLUDE entity-name] ...]]

[ENTITY entity-name [,entity-name] ...]

RULES:

1. The keyword EXPLICIT is used to denote entities which are not implicit. If the first subclause is omitted, the default condition is EXPLICIT.
2. If only implicit entities are to be selected, then the keyword IMPLICIT is used and the ENTITY-TYPE and ENTITY clauses must not be included.
3. If the user wishes to have both, but to receive the list in two parts -- explicit followed by implicit -- then the keyword BOTH is used. The ENTITY-TYPE and ENTITY clauses are then only meaningful to the explicit entities in the dictionary.
4. entity-type-name must be the name of an entity-type in the schema.
5. All entity-names in the EXCLUDE part must be of the same type as the entity-type-name. The effect of the ENTITY-TYPE clause is thus to qualify all of the

entities that have a given entity type-name, but to EXCLUDE those specifically stated; if any entity-name stated in the qualification statement does not exist (e.g., if the clause is part of a saved procedure that referred to a since-deleted entity) its effect is ignored but a warning message is given (i.e., it does not cause an error, but it will generate a message to the user).

ACTIONS PERFORMED:

1. If only EXPLICIT, IMPLICIT, or BOTH is given, then the selection is based on that criteria only. The use of IMPLICIT means that no other subclause is meaningful.
2. BOTH is equivalent to ALL' (in the description of the QUALIFY command) except that the use of all may produce a merged list (depending on the implementation method of the vendor).
3. The use of ENTITY-TYPE is to qualify all entities of that named type. Then entities may be excluded from this list.

ERROR CONDITIONS:

1. The IMPLICIT keyword has been given with another subclause.
2. entity-type-name is not the primary name of an entity-type in the schema.
3. entity-name is not of the type entity-type-name.

EXAMPLES

1. QUALIFY IMPLICIT

This makes the current-list a list of all implicitly defined entities.

2. QUALIFY ENTITY-TYPE FILE

This gives a list of all primary names of entities of type file in the current-list.

3. QUALIFY ENTITY-TYPE FILE EXCLUDE PERSONNEL-FILE

This gives the same list as in Example 2. above, except that PERSONNEL-FILE will not be included.

4. QUALIFY ENTITY PERSONNEL-FILE, STUDENT RECORD

This will produce a current-list of the two entities named.

6.2.1.2 ALTERNATE-NAME-SELECTION CLAUSE

PURPOSE: IDENTIFICATION-NAMES is an attribute-group-type that consists of the attribute-type ALTERNATE-NAME and ALTERNATE-NAME-CONTEXT. This clause selects one or more dictionary primary names that correspond to an ALTERNATE-NAME.

FORMAT: [[[ENTITY alternate-name] [ALTERNATE-NAME-CONTEXT = alternate-name-context]

[FOR ENTITY-TYPE entity-type-name ONLY]] ...

RULES:

1. This format with no context or entity-type subclauses is indistinguishable from the entity-selection-clause.

It provides all entity-names that correspond to a given alternate-name.

2. The alternate-name context subclause allows restriction based on a specific context (e.g., ALTERNATE-NAME-CONTEXT = COBOL). If only this is specified, all alternate-names with this context are converted to their primary name (e.g., primary names of all entities that have COBOL names).
3. The entity-type subclause restricts to a particular type (e.g., for FILE only); the use of this with a context subclause allows further restriction (e.g., ALTERNATE-NAME-CONTEXT = FORTRAN FOR ENTITY-TYPE ELEMENT ONLY gives a qualification list of entities that are elements and have a FORTRAN name).
4. The FOR subclause cannot be the only subclause; i.e., it must restrict either an entity or alternate-name context subclause.
5. If the given alternate-name or alternate-name-context is not found in the dictionary, the system returns a null list as the contribution of this subclause.
6. The FOR subclause is not allowed without either ENTITY or ALTERNATE-NAME-CONTEXT subclause (or both).
7. The entity-type-name must be a name of an entity-type in the schema.

ACTIONS PERFORMED:

1. The list of entities that are qualified are selected based on the alternate-name(s) given. The list will contain only primary names of these entities.
2. The context will be used to select the list of entities or else to restrict those alternate-names (e.g., if two different contexts use the same name, only one would be selected).
3. The use of the FOR ENTITY-TYPE subclause will also

further restrict the selection.

ERROR CONDITIONS:

1. The alternate-name-context and entity alternate-name subclauses are not given though a FOR clause is specified.
2. There is no entity that qualifies with the type named in the FOR subclause.

EXAMPLES

1. QUALIFY ENTITY PERSONNEL-FILE ENTITY ZAP-FILE

Assuming here that PERSONNEL-FILE is a primary name and that ZAP-FILE is an alternate-name of MACHINE-FILE, then the current-list becomes (PERSONNEL-FILE, MACHINE-FILE).

2. QUALIFY ALTERNATE-NAME-CONTEXT = COBOL

The current-list becomes a list of the primary names of all entities that have a COBOL context.

3. QUALIFY ZIPS ENTITY ZIP

Assuming that the COBOL name of the entity with primary name PATH is ZIP, and that the FORTRAN name of the entity with primary name PAGE is also ZIP, then the result is a qualification list with name ZIPS with members PATH and PAGE.

4. QUALIFY ZIPS ENTITY ZIP ALTERNATE-NAME-CONTEXT = COBOL

The result is as in 3. above, but the list only contains PATH.

5. QUALIFY ZIPS ENTITY ZIP FOR ENTITY-TYPE LOCATION ONLY

Assuming that LOCATION is an entity-type, and that PATH is an entity of that type, whereas PAGE is not,

then ZIPS contains only PATH.

6.2.1.3 PRIMARY-NAME-RESTRICTION CLAUSE

PURPOSE: To restrict the selection of qualified entities to those with certain characteristics in their primary names, a clause is provided which allows the selection of entities whose primary names contain certain character strings (such as "DATE") or start/end with certain strings (such as "FIRST-"/"-NUMBER" respectively), or combinations of these. It also allows selection of a set that starts with one character string and ends with another (such as all entity-names from PERSON- to PERSONNEL-). The length requirements may also be specified.

The character "*" is used as a "don't care" character (i.e., a character that always matches any other character).

FORMAT: [STARTS AS "string (with * as don't care characters)"]
[ENDS AS "string (with * as don't care characters)"]
[CONTAINS "string (with embedded *)"]
[BETWEEN "string" AND "string"]
[LENGTH OP integer]

where OP is one of the following:

=	(equal to)
>	(greater than)
<	(less than)
≠	(not equal to)
→	(not greater than, i.e. less than or equal to)
←	(not less than, i.e. greater than or equal to)

RULES:

1. The string must not contain characters that are not allowed in the naming of entities.
2. The use of BETWEEN implies that there is an order and that the first string precedes the second in that order.

ACTIONS PERFORMED:

1. The primary names of entities that were selected by previous clauses are examined on the basis of whether they start with, end with, or contain a given string, or are within a certain group of characters. Alternately, their lengths are examined to see if they qualify.
2. The restricted set of entities is then processed by the next clause if one exists; otherwise it is output.

ERROR CONDITIONS:

1. The string given is invalid, in that it contains characters not allowed in a name (e.g., a carriage return).
2. The two strings in the BETWEEN subclause are not in the correct order.
3. The length is not specified as an integer.

EXAMPLES

1. QUALIFY ALL STARTS AS "PERSON"

The current-list will contain all entities whose primary name start with PERSON.

2. QUALIFY ZIP ENTITY-TYPE FILE CONTAINS "F*R"

This command creates a qualification list containing all files whose primary name contain the character F followed by some other character and then followed by the character R. For example, the list might contain FORWARD-FILE, FILE-FAR-PLACES, FIRST-FIELD-FILE, INFERRED-FILE, UNFORMATTED-FILE.

3. QUALIFY ZIP ENTITY-TYPE FILE CONTAINS "F*R"
BETWEEN "FIL" AND "FIR"

In the above example, the list would contain FILE-FAR-PLACES, FIRST-FIELD-FILE.

4. QUALIFY ZIP ENTITY-TYPE FILE CONTAINS "F*R"
LENGTH > 14

Again referring to the previous example, the list would contain FILE-FAR-PLACES, FIRST-FIELD-FILE, UNFORMATTED-FILE.

6.2.1.4 RELATIONSHIP-RESTRICTION CLAUSE

PURPOSE: It may be necessary to restrict the selection of any entity based on the relationship(s) or attributes of relationship(s) in which it participates. Since a relationship has an inverse name, the user will need to select that relationship-type-name or relationship-class-type-name which uses the entity-type-name as the start of the relationship or relationship class.

FORMAT: [WHERE ENTITY IS START OF
{relationship-type-name|relationship-class-type-name|relationship-type-chain-name|ALL RELATIONSHIPS}
ENDING WITH {primary-name|alternate-name|
entity-type-name}[AND RELATIONSHIP(s){HAS|HAVE}
conditional-attribute-subclause[,conditional-
attribute-subclause]...]]

where conditional-attribute-subclause is made up of an attribute-restriction-clause:

[NOT] conditional-subclause [{AND|OR}[NOT]]

where conditional-subclause is:

attribute-type-name {{OP}value-1|BETWEEN value-2
AND value-3}

where OP is one of the following:

=	(equal to)
>	(greater than)
<	(less than)
≠	(not equal to)
→	(not greater than, i.e. less than or equal to)
←	(not less than, i.e. greater than or equal to)

RULES:

1. Any relationship-type-name or relationship-class-type-name or relationship-type-chain-name must refer to an entity-name that has been previously selected and retained by the previous clauses of the qualification statement.
2. A relationship-type-chain is a sequence

relationship-type-1, ... , relationship-type-n

where consecutive relationships in the sequence have a common entity-type, and where loops are not permitted. All entities that are members of a relationship whose relationship-type is a component of a chain will be reported on, and all attributes and attribute-groups of the relationship-types will be shown. relationship-type-1 must have as a member an entity-type which pertains to the entities being reported on.

3. Any primary-name or alternate-name given must refer to

an entity-type that may be connected to one or more of the entity-name (as selected by previous clauses) in the named relationship(s) or chains of relationships. The same is true for any entity-type-name given in this subclause.

4. If a conditional-attribute-subclause is used, any named attribute-types and their corresponding values must be meaningful with respect to one of the relationships (i.e., they must refer to attributes of one of the possible relationships).
5. The conditional-attribute-subclause may have several conditional subclauses linked by the logical operators NOT, AND, and OR. The precedence of the unary operator NOT is highest. Then the precedence of the binary operators is that AND is higher than OR. Parentheses may be included to change the precedence.
6. value-1 of a BETWEEN option must be less than value-2. The use of attribute-types that are alphabetic rather than numeric is possible. Then the order is implied as Roman alphabetic order. Mixed character (numeric and alphabetic) values are not allowed.
7. AND, OR, and NOT have their normal mathematical meaning.

ACTIONS PERFORMED:

1. The set of entities that have been selected and retained so far are in turn assumed to be the first entity of the various named relationships, etc. The second entity of the relationship is then matched with the ENDING WITH entities. If the relationship exists, the originally selected entity is retained, otherwise it is rejected.
2. If all relationships are tested and the second entity could not be matched, an error condition is given.
3. If a RELATIONSHIP(s) HAS or HAVE subclause is used, the attributes of all relationships with matching

entities are tested to see whether they have the necessary values, or come within the range, or are larger, or higher in the alphabet, etc. Only those first entities that comply with this condition are retained.

4. If there is no way that a condition can be satisfied (e.g., this attribute-type does not exist for any of the relationships that are available) then an error condition occurs.

ERROR CONDITIONS:

1. The "start" entity-types or entities (that have been selected due to previous qualification clauses) are not meaningful to at least one of the possible relationships; the entity is removed from the qualification list and the user informed of a probable error condition.
2. The entity-types or entities in the ending clause are not meaningful to one of the possible named relationships; the qualification aborts with an error message to the user.
3. An attribute-type is not meaningful to at least one of the possible named relationships; the qualification aborts with an error message to the user.
4. The named attribute-type does not exist in the schema.
5. The values in a BETWEEN subclause are not in order.
6. The values given do not comply with the representation (PICTURE) of the attribute-type (e.g., they are numeric while it is defined as alphabetic).

EXAMPLES

1. QUALIFY ALL WHERE ENTITY IS START OF PROCESSED-BY
ENDING WITH PERSON-PROCEDURE

This gives a current qualification list of each entity that is processed by the entity PERSON-PROCEDURE. The relationship-class-type PROCESSED-BY exists between data and process entities. Thus the only results (in the current list) will be data entities that are processed by the particular procedure PERSON-PROCEDURE.

2. QUALIFY ENTITY-TYPE FILE CONTAINS "F*R" WHERE ENTITY
IS START OF HAS-SORT-KEY ENDING WITH SOCIAL-SECURITY-
NUMBER

This produces a current list that is the result of the second example in 6.2.1.3, except that only those that have the identifier (key) of social security number (as a whole or part key) will be retained (i.e., the previous list has been further restricted).

3. QUALIFY ENTITY-TYPE FILE WHERE ENTITY IS START OF HAS-
SORT-KEY ENDING WITH SOCIAL-SECURITY-NUMBER AND
RELATIONSHIP HAS ALTERNATE-NAME-CONTEXT = "COBOL"

This qualifies files with a sort key of social security number and that have a designated relationship context of COBOL (i.e., it can be used in a COBOL context).

6.2.1.5 AUDIT-ATTRIBUTE-RESTRICTION CLAUSE

PURPOSE: The system standard schema contains a number of attribute-types that are maintained by the system. Among these are the date and person who created the entity, the date at which it was last modified and the name of the modifier, and the total number of modifications. The selection of entities will be based on attributes

of these types.

FORMAT: [CREATED [{BETWEEN date AND date|ON date|
 BEFORE date|AFTER date}][BY person-name]]

 [LAST-CHANGED [{BETWEEN date AND date|ON date|
 BEFORE date|AFTER date}][BY person-name]]

 [TOTAL-CHANGES OP integer]

where OP is one of the following:

=	(equal to)
>	(greater than)
<	(less than)
≠	(not equal to)
→	(not greater than, i.e. less than or equal to)
←	(not less than, i.e. greater than or equal to)

RULES:

1. The time, or person, or number-of-changes, or any or all of these may be used to qualify an entity.
2. If the BETWEEN subclause is used, the first date must be earlier than the second.
3. Any or all of the above subclauses may be given. They are ANDed together.

ACTIONS PERFORMED:

1. The previously selected entities (from other clauses) are examined to see whether they comply with the condition or conditions. The restricted list (i.e., those that comply with the "audit" restrictions) is then passed on to the next clause or for output.

ERROR CONDITIONS:

1. Dates in the BETWEEN subclause are incompatible.
2. The given person-name does not exist in the dictionary.
3. An integer is not specified in the TOTAL-CHANGES clause.

EXAMPLES

1. QUALIFY ALL CREATED ON 24/MAY/1982

This command produces a current-list of all entities that were created on that date.

2. QUALIFY ENTITY-TYPE FILE CREATED BY JOHNSON

This command produces a current-list of all files that Johnson created.

3. QUALIFY ENTITY-TYPE FILE CREATED BY JOHNSON LAST-CHANGED AFTER 24/MAY/1982

4. QUALIFY ENTITY-TYPE FILE CONTAINS "F*R"
TOTAL-CHANGES > 5

This command produces a current-list of files with the string F don't care R (as in 6.2.1.3) but only if changed more than 5 times.

6.2.1.6 ATTRIBUTE-RESTRICTION CLAUSE

PURPOSE: Attributes may be used to further restrict the selection of an entity. Also, because there may be several attribute-types of interest, this clause uses Boolean expressions joined by conjunction (AND) and disjunction (OR), as well as negation (NOT).

FORMAT: [[NOT] conditional-subclause [{AND|OR}[NOT]
 conditional-subclause] ...]

where conditional-subclause is:

attribute-type-name {{OP}value-1|BETWEEN value-2
 AND value-3}

where OP is one of the following:

=	(equal to)
>	(greater than)
<	(less than)
≠	(not equal to)
→	(not greater than, i.e. less than or equal to)
←	(not less than, i.e. greater than or equal to)

RULES:

1. If the attribute-type does not exist in the dictionary, no entity will be qualified.
2. The attribute-restriction-clause may have several conditional subclauses linked by the logical operators NOT, AND, and OR. The precedence of the unary operator NOT is highest. Then the precedence of the binary operators is that AND is higher than OR. Parentheses may be included to change the precedence.
3. value-1 of a BETWEEN option must be less than value-2. The use of attribute-types that have alphabetic rather than numeric values is possible. Then the order is implied as Roman alphabetic order. Mixed character (numeric and alphabetic) values are not allowed.
4. AND, OR, and NOT have their normal mathematical meaning.

ACTIONS PERFORMED:

1. The set of previously selected entities are now further restricted by checking the values of their attributes. Only those entities that match the criterion of their attributes are retained.

ERROR CONDITIONS:

1. The attribute-type does not exist in the dictionary.
2. The values in the BETWEEN subclause are not in order.
3. The values do not have the same type (e.g., numeric or alphabetic) as the attribute type.

EXAMPLES

1. QUALIFY ENTITY-TYPE FILE STATUS = CONTROLLED

This produces a current-list of files that are in controlled status.

2. QUALIFY ZIP ENTITY-TYPE MANAGEMENT
STATUS = CONTROLLED AND STAGE = ULTIMATE
OR POLICY-CONDITION = PRESIDENT-SCAN

This produces the list, in ZIP, of entities of type MANAGEMENT that are either in the ultimate stage and in a controlled status, or else have a policy condition of "president-scan". Note that since the precedence of AND is higher than OR, this is the same as:

QUALIFY ZIP ENTITY-TYPE MANAGEMENT
POLICY-CONDITION = PRESIDENT-SCAN OR
STAGE = ULTIMATE AND STATUS = CONTROLLED

6.2.1.7 TEXT-STRING-RESTRICTION CLAUSE

PURPOSE: This clause allows restriction of previously qualified entities based on some characteristic in their DESCRIPTION or COMMENTS attribute-type or some other attribute of an attribute-type with meta-attribute PICTURE = TEXT.

FORMAT: [subclause [{AND|OR} subclause] ...]

where subclause is:

attribute-type-name [NOT] condition [{AND|OR}
[NOT] condition] ...

and condition is one of the following:

STARTS AS "string (with * as don't-care)"
ENDS WITH "string (with * as don't-care)"
CONTAINS "string (with embedded *)"

RULES:

1. attribute-type-name must be the name of an attribute-type for which PICTURE = TEXT.
2. AND has a higher precedence than OR.
3. The use of NOT clauses can cause long searches; thus the user should be warned about the potential high cost of its use.

ACTIONS PERFORMED:

1. The previously selected list of entities is examined and any that do not satisfy the condition are discarded.

ERROR CONDITIONS:

1. The string contains invalid characters for names.

EXAMPLES

1. QUALIFY ALL DESCRIPTION CONTAINS "PAYROLL"
AND COMMENT STARTS AS "FIRST,"

This command takes all entities that have the word "payroll" in their description and have a comment that has "first" followed by a comma at the start, and puts them in the current-list.

2. QUALIFY ZIP ENTITY-TYPE FILE CREATED BY JOHNSON
DESCRIPTION CONTAINS "PAYROLL"

This command generates a list named ZIP with entities of type file created by Johnson and containing "payroll" in their description.

6.2.1.8 CLASSIFICATION-RESTRICTION-CLAUSE

PURPOSE: This clause allows further restriction of entities based on classification keywords which are the attributes of the attribute-type CLASSIFICATION. All entities have such attributes.

FORMAT: CLASSIFICATION-KEYWORD = [NOT] keyword [{AND|OR}
[NOT] keyword] ...

RULES:

1. AND has a higher precedence than OR.
2. The use of NOT clauses can cause long searches; thus the user should be warned on the potential high cost of its use.

ACTIONS PERFORMED:

1. The previously selected entities are examined to determine whether they comply with the required classification. If they do not, they are rejected, otherwise they are ready for output.

ERROR CONDITIONS:

1. None.

EXAMPLES

1. QUALIFY ENTITY-TYPE FILE
CLASSIFICATION-KEYWORD = PAY AND NOT PERSONNEL

This produces a current-list of files that have a keyword of PAY but do not also have a keyword of PERSONNEL.

2. QUALIFY ENTITY-TYPE FILE
CREATED BY JOHNSON
DESCRIPTION CONTAINS "PAYS"
CLASSIFICATION-KEYWORD = LIBRARY

This provides the current-list of files created by Johnson with a description that contains the string "pay" and has been classified with a keyword "library".

6.2.2 UNION, INTERSECTION, SET-DIFFERENCE COMMANDS

PURPOSE: The normal set operators: union, intersection, and set difference can be applied to two qualification lists to produce a new qualification list. Since one can either name a list, or generate it, or use a current-list, there are several forms of these commands.

FORMAT: {UNION|INTERSECTION|SET-DIFFERENCE}

([qualification-list-name|qualification-command|RUN-command]),

{[qualification-list-name]|qualification-command|RUN-command},

[qualification-list-name])

RULES:

1. Each command has three parameters. The first must either be:
 - o the name of a qualification list that already exists;
 - o or a qualification statement that can be evaluated to obtain a qualification list; or
 - o a RUN command (as discussed in 6.2.5) applied to a saved qualification procedure -- which also evaluates to a qualification list.

In the latter two cases, the result is returned to the command and the current-list is not modified. If the parameter is missing, the current-list is used as the missing parameter.

2. The second parameter may be missing, in which case,

the current-list is used as the missing parameter. Thus the second parameter is the same as the first.

3. In case of any error condition, the user is informed and the command aborted.

ACTIONS PERFORMED:

1. The set operation specified is performed on the first two parameters. The resulting qualification list is then given the name in the third parameter without altering the current-list, or if the third parameter is missing, the resulting list becomes the current-list.
2. A count of entity-names in the resulting qualification list is provided.

ERROR CONDITIONS:

1. The RUN-command does not refer to a saved process.
2. The qualification-list-name in arguments 1 or 2 is not known to the system.

EXAMPLES

1. UNION RUN ZIP-PROC,

This command runs the procedure ZIP-PROC, then unions the result with the current-list and puts the result as the current-list.

2. INTERSECTION, NAME-1,

This command uses the stored list (NAME-1) and intersects it with the current-list, giving a new current-list as the result.

3. SET-DIFFERENCE RUN ZIP-PROC, NAME-1, NAME-1

This command takes NAME-1 from the results of running the ZIP-PROC and puts the new results in NAME-1.

6.2.3 DISQUALIFY-LIST COMMAND

PURPOSE: The deletion of any current-list occurs at the end of a user session. Any current-list is replaced after a new current-list has been generated. The lists that have been generated and named during a session will be retained until the end of the session, but any one may be deleted by use of this command.

FORMAT: DISQUALIFY-LIST qualification-list-name

RULES:

1. If the qualification-list-name is not known to the system, the user is informed and the command aborted. Otherwise the list is deleted, with all references to it.

ACTIONS PERFORMED:

1. The named qualification list is deleted from the system.

ERROR CONDITIONS:

1. The named list does not exist.

EXAMPLES

1. DISQUALIFY-LIST LIST-NAME

The qualification list LIST-NAME is deleted.

6.2.4 SAVE-LIST COMMAND

PURPOSE: In order to allow qualifications that are made during one session to be available in the next, a procedure for generating it must be saved. Because dictionary changes may occur between sessions, it is unsafe to save the list of entities; thus this command saves the process of generating the list. However, since the list of interest may have been the result of several qualification commands, the stored procedure is not necessarily a simple (single) command.

FORMAT: SAVE-LIST {qualification-list-name|CURRENT-LIST}
AS procedure-name
[SAVE-TEXT(text)]

RULES:

1. The object process to be saved may either be one that generated the named list or the current-list.
2. The process is stored (and subsequently retrieved) under the procedure-name; procedure-name cannot be the name of an existing procedure.
3. The user may supply some special text or commands that may be printed out later.
4. Any saved text may be retrieved by:

GIVE-MAVED-TEXT procedure-name

5. The system determines what string of commands are

necessary to generate the procedure; the method is implementation-dependent.

ACTIONS PERFORMED:

1. The user issues the command naming a stored list or the current-list.
2. The system then determines how this list was generated (possibly from the audit trail or by having previously stored a procedure in the system every time a list is generated or changed).
3. The procedure is stored under the given name at some location that is implementation dependent.

ERROR CONDITIONS:

1. qualification-list-name is not the name of a qualification list.
2. There exists a procedure with the specified name.

EXAMPLES

1. SAVE-LIST CURRENT-LIST AS ZIP-PROC
SAVE-TEXT (THIS IS ALL FILES)

This command saves the procedure needed to generate the current-list, and the text portion of the procedure.

2. SAVE-LIST CITY AS CITY-PROC

Here the procedure to generate list CITY is saved.

6.2.5 RUN COMMAND

PURPOSE: Once a procedure for generation of a qualification list has been saved, it may be executed by a RUN command. This must be given the name by which the procedure was stored and, optionally the name of the generated list.

FORMAT: RUN procedure-name [GIVING qualification-list-name]

RULES:

1. The command gives a response of the number of entries in the qualified list of entity names.
2. When free standing, this command generates either a list named as stated in the GIVING option or else produces a new current-list.
3. When embedded in a qualification list manipulation command (UNION, INTERSECTION, OR SET DIFFERENCE; see 6.2.2) or in query and reporting, the current-list is never changed during the execution of the RUN command (i.e. the result, if unnamed takes part in the set operation but is not recorded elsewhere).
4. If the procedure-name does not exist, the user is informed and the command aborts.

ACTIONS PERFORMED:

1. The procedure is run to give either a new qualification list or the named list.

ERROR CONDITIONS:

1. procedure-name does not exist.

EXAMPLES

1. GIVE-~~SAVED~~-TEXT ZIP-PROC
RUN ZIP-PROC GIVING ZIP

This command uses the saved text (see 6.2.4) to check that it is the required procedure and then generates the new ZIP list using ZIP-PROC.

2. RUN ZIP-PROC

Here the current-list is the result of running the procedure.

6.2.6 DROP-PROCEDURE COMMAND

PURPOSE: To delete a previously saved procedure that could be used for generating a qualification list.

FORMAT: DROP-PROCEDURE procedure-name

RULES:

1. The named procedure is deleted from the system.
2. If the name is not known to the system, the user is informed and the command aborted.

ACTIONS PERFORMED:

1. The procedure named is no longer available to the user.

ERROR CONDITIONS:

1. The procedure-name is not known in the dictionary.

EXAMPLES

1. DROP-PROCEDURE ZIP-PROC

The procedure ZIP-PROC is deleted from the system.

6.2.7 LIST-QUALIFICATIONS COMMAND

PURPOSE: To give a list of all stored qualification procedures.

FORMAT: LIST-QUALIFICATIONS
[TEXT ALSO]
[PROCEDURE ALSO]

RULES:

1. A count is appended to the output list.
2. Optionally any text can also be printed.

ACTIONS PERFORMED:

1. A list of all stored qualifications is output.
2. Any text is given with the name, if requested.
3. The original stored material is then optionally given.
4. The count is then given.

ERROR CONDITIONS:

1. None.

6.3 MAINTENANCE COMMANDS

The following maintenance commands will be specified:

ADD-ENTITY
ADD-RELATIONSHIP
CHANGE-STATUS
COPY
DECLARE
DELETE-ENTITY
DELETE-RELATIONSHIP
MODIFY-ENTITY
MODIFY-RELATIONSHIP
RENAME
RENUMBER

The order used is alphabetic by command name.

6.3.1 ADD-ENTITY COMMAND

PURPOSE: To add an entity in the dictionary.

FORMAT: ADD-ENTITY
 entity-type-name [entity-name]
 [clause-1 [,clause-2] ...]
 [security clause]

 where security clause is

 {NEW-CONTROLLER|EXISTING-CONTROLLER}
 controller-name

RULES:

1. entity-type-name must be the name of an entity-type in the dictionary schema.

2. Whether or not entity-name is specified depends on whether or not the entity-type specified has system-generated primary names.

(a) For an entity-type which does not have system-generated names, entity-name must be specified.

(b) For an entity-type which does have system-generated names, entity-name must not be specified.

3. entity-name cannot be the primary name of an entity in the dictionary.

4. entity-name is subject to the entity-name rules described in 6.1.1.

5. clause-1 ... are attribute or attribute-group clauses.

6. An attribute clause is of the form

attribute-type-name = attribute-1 [,attribute-2] ...

7. If an attribute-type has PICTURE = TEXT, the form of the clause is as discussed in 6.1.3.

8. For an attribute-group-type with name attribute-group-type-name which is composed of the attribute-types with name attribute-type-name-1, ..., attribute-type-name-N an attribute-group clause is of the form

attribute-group-type-name =
 (attribute-1-1, ..., attribute-N-1)
 [, (attribute-1-2, ..., attribute-N-2)] ...

where attribute-1-j is an attribute of attribute-type-name-1, ...

9. Every attribute clause must correspond to an attribute-type that pertains to the entity-type with name entity-type-name (i.e. there exists a meta-

relationship in the dictionary schema with the given entity-type and the attribute-type being used as members).

10. Every attribute-group clause must correspond to an attribute-group-type that pertains to the entity-type with name entity-type-name (i.e. there exists a meta-relationship in the dictionary schema with the given entity-type and the attribute-group-type being used as members).
11. If an attribute-type belongs to an attribute-group-type, attributes of that type can only be specified in the context of the attribute-group. Hence, if there is an attribute-group-type that pertains to the given entity-type, none of the attribute-types that make up the attribute-group-type can appear in an attribute clause.
12. Every attribute in an attribute clause or an attribute-group clause is subject to the attribute rules described in 6.1.3.
13. Attribute clauses for attribute-types which are system-generated cannot be specified.
14. If EXISTING-CONTROLLER is specified in the security-clause, controller-name must be the primary name of an existing entity of type ACCESS-CONTROLLER.
15. If NEW-CONTROLLER is specified in the security-clause, controller-name cannot exist in the dictionary as the primary name of an entity.

ACTIONS PERFORMED:

1. The entity with name entity-name is added to the dictionary.
2. Attributes and attribute-groups specified in attribute clauses and attribute-group clauses are assigned to this entity.

3. Values are inserted by the system for the audit attribute-types.
4. If NEW-CONTROLLER is specified in the security-clause, an entity of entity-type ACCESS-CONTROLLER with primary name controller-name is created in the dictionary.
5. If a security-clause has been specified, the entity being added is protected by the ACCESS-CONTROLLER that has been specified.
6. The command is recorded in the log/audit file.
7. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. entity-type-name is not the name of an entity-type in the dictionary schema.
2. The entity-type specified does not have system-generated primary names and no entity-name is specified.
3. The entity-type specified has system-generated primary names and an entity-name is specified.
4. There exists an entity in the dictionary with primary name entity-name.
5. entity-name violates a rule about primary names of entities of the specified entity-type.
6. An attribute clause is specified where the attribute-type does not pertain to the specified entity-type.
7. An attribute-group clause is specified where the attribute-group-type does not pertain to the specified entity-type.
8. An attribute clause is specified which is not in the prescribed form.

9. An attribute-group clause is specified which is not in the prescribed form.
10. An attribute-type is specified in an attribute clause which is part of an attribute-group-type for the specified entity-type.
11. An attribute is specified which violates the attribute rules described in 6.1.3.
12. An attribute clause is specified for an attribute-type whose values are system-generated.
13. EXISTING-CONTROLLER has been specified in a security-clause and controller-name is not the primary name of an existing entity of type ACCESS-CONTROLLER.
14. NEW-CONTROLLER has been specified in a security-clause and controller-name is the primary name of an entity in the dictionary.
15. More than one security-clause has been specified.

EXAMPLE

1. ADD-ENTITY
PROGRAM PAYROLL-PROGRAM
DESCRIPTION = "THIS PROGRAM IS USED TO PREPARE THE
WEEKLY PAYROLL."
FREQUENCY = WEEKLY
NUMBER OF LINES OF CODE = 3000
LANGUAGE = COBOL
EXISTING-CONTROLLER = AC4097
2. ADD-ENTITY
DOCUMENT FORM-A243
STATUS = PROPOSED
ORIGINATOR = DEPT-B54
DOCUMENT-TYPE = FORM

6.3.2 ADD-RELATIONSHIP COMMAND

PURPOSE: To create relationships in the dictionary. One of the two entities which are members of the relationship created must exist in the dictionary, and the command will create the other entity if not existent prior to the command. There exist two forms of the command, one using the name of a relationship-type in the command, and the other one using the name of a relationship-class-type. The essential difference between these two forms of the command is that whenever a relationship-type is specified the entity-types of the two entities participating in the relationship are uniquely determined, whereas when the name of a relationship-class-type is used, in certain cases there is an ambiguity in the entity-type of one of the entities. In the case where the entity-type of the non-existent entity is either specified or can be determined, this entity is created as if an ADD-ENTITY command had been given. In the case where the entity-type of this entity is not specified or cannot be determined this entity is created in the dictionary as an implicit entity. Once an implicit entity has been created, the only maintenance commands that can address it are the DECLARE command, which is used to specify the entity-type, and the DELETE-ENTITY and DELETE-RELATIONSHIP commands, which can delete an implicit entity.

FORMAT: Form 1 - using relationship-type

```
ADD-RELATIONSHIP
relationship-type-name
entity-name-1, entity-name-2
[SEQUENCE PARAMETER = parameter]
[clause-1 [,clause-2] ... ]
```

Form 2 - using relationship-class-type

```
ADD-RELATIONSHIP
VIA RELATIONSHIP-CLASS
relationship-class-type-name
```

```
[entity-type-name-1] entity-name-1,  
      [entity-type-name-2] entity-name-2  
[SEQUENCE PARAMETER = parameter]  
[clause-1 [,clause-2] ... ]
```

RULES:

1. There must exist an entity in the dictionary with primary name either entity-name-1 or entity-name-2. This entity will be referred to here as the "existing entity". If there exist two entities in the dictionary with primary name entity-name-1 and entity-name-2, respectively, each one of these entities will be referred to as an "existing entity".
2. An "existing entity" must not be an implicit entity.
3. In Form 2 of the command, if the optional entity-type-name is specified for an "existing entity", it must be the name of the entity type of that entity.
4. In Form 2 of the command, if entity-name-1 is not the name of an "existing entity" (and hence, because of Rule 1, entity-name-2 is the name of an "existing entity"), if entity-type-name-1 is specified, this entity-type must be consistent with the possible entity-types that can be members of the specified relationship-class-type.
5. In Form 2 of the command, if entity-name-2 is not the name of an "existing entity" (and hence, because of Rule 1, entity-name-1 is the name of an "existing entity"), if entity-type-name-2 is specified, this entity-type must be consistent with the possible entity-types that can be members of the specified relationship-class-type.
6. In Form 2 of the command, if
 - a) the entity with primary name entity-name-1 is not an "existing entity", and that entity-type-name-1 is not specified, and

- b) that furthermore, for the given relationship-class-type the entity-type of the entity with primary name entity-name-1 is not uniquely determined,

it then follows that the entity-type of the entity with primary name entity-name-1 cannot be deduced by the system and that this entity must be added into the dictionary as an implicit entity.

7. In Form 2 of the command, if

- a) the entity with primary name entity-name-2 is not an "existing entity", and that entity-type-name-2 is not specified, and
- b) that furthermore, for the given relationship-class-type the entity-type of the entity with primary name entity-name-2 is not uniquely determined,

it then follows that the entity-type of the entity with primary name entity-name-2 cannot be deduced by the system and that this entity must be added into the dictionary as an implicit entity.

- 8. In Form 2 of the command, if either the entity with primary name entity-name-1 or entity-name-2 is to be added as an implicit entity, the command must not contain any of the clauses clause-1,
- 9. The relationship being added cannot already exist in the dictionary.
- 10. If a SEQUENCE PARAMETER clause is specified, it must apply to the relationship-type an instance of which is being added.
- 11. clause-1 ... are attribute or attribute-group clauses.
- 12. An attribute clause is of the form

attribute-type-name = attribute-1 [,attribute-2] ...

13. If an attribute-type has PICTURE = TEXT, the form of the clause is as discussed in 6.1.4.
14. For an attribute-group-type with name attribute-group-type-name which is composed of the attribute-types with name attribute-type-name-1, ..., attribute-type-name-N an attribute-group clause must be of the form

attribute-group-type-name =
 (attribute-1-1, ..., attribute-n-1)
 [, (attribute-1-2, ..., attribute-n-2)] ...

where attribute-1-j (j=1, ... ,m) is an attribute of attribute-type-name-1.

15. Every attribute clause must correspond to an attribute-type that pertains to the relationship-type with name relationship-type-name (i.e. there exists a meta-relationship in the dictionary schema with the given relationship-type and the attribute-type being used as members).
16. Every attribute-group clause must correspond to an attribute-group-type that pertains to the relationship-type with name relationship-type-name (i.e. there exists a meta-relationship in the dictionary schema with the given relationship-type and the attribute-group-type being used as members).
17. A value of an attribute-type that is a member of an attribute-group-type that pertains to the given relationship-type can only be specified in the context of an attribute-group-type clause.
18. Every attribute in an attribute clause or an attribute-group clause is subject to the attribute rules described in 6.1.3.
17. Attribute clauses for attribute-types which are system-generated cannot be specified.
18. Rules for specification of attribute-groups of attribute-group-type USAGE-NAMES are as given in 6.1.5.

19. Names used for relationship-types and relationship-class-types can either be forward names (i.e. the primary names) or inverse names.

ACTIONS PERFORMED:

1. If entity-name-1 or entity-name-2 is the primary name of an entity which does not exist in the dictionary, and if the entity-type of this entity either is given or can be determined uniquely, this entity is entered into the dictionary as an instance of this entity-type.
2. In the above case or in the case where both entities exist in the dictionary, a relationship of the specified relationship-type is established whose members are the entities with primary name entity-name-1 and entity-name-2. The specified attributes and attribute-groups are assigned to this relationship.
3. If the entity with primary name entity-name-1 exists in the dictionary, and if
 - a) the entity with primary name entity-name-2 does not exist in the dictionary, and
 - b) the entity-type of this entity is not specified or cannot be determined uniquely,the entity with primary name entity-name-2 is entered into the dictionary and it is noted that a relationship of undetermined relationship-type exists between the entities with primary names entity-name-1 and entity-name-2.
4. If the entity with primary name entity-name-2 exists in the dictionary, and if
 - a) the entity with primary name entity-name-1 does not exist in the dictionary, and
 - b) the entity-type of this entity is not specified or cannot be determined uniquely,

the entity with primary name entity-name-1 is entered into the dictionary and it is noted that a relationship of undetermined relationship-type exists between the entities with primary names entity-name-1 and entity-name-2.

5. Proper values are inserted into the audit attributes of the relationship, and if an entity is added, of that entity.
6. The command is recorded in the log/audit file.
7. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. Neither of the entities with names entity-name-1 or entity-name-2 exists in the dictionary.
2. One of the entities specified is an implicit entity.
3. An invalid entity-type is specified for an "existing entity".
4. An implicit entity is to be created in the dictionary and an attribute-clause or attribute-group-clause has been specified.
5. A required SEQUENCE PARAMETER clause has been omitted.
6. The relationship to be added already exists in the dictionary.
7. An attribute-clause or attribute-group-clause is specified that does not pertain to the relationship being created.
8. An attribute-type appears in an attribute-clause that should be specified in an attribute-group-clause.
9. An attribute is specified which is not valid for the

specified attribute-type.

10. An attribute is specified which is not valid for the specified attribute-group in which it appears.
11. An entity is to be added which does not conform to the entity-name rules.

EXAMPLES

1. Let A-SYSTEM be the name of an entity of entity-type SYSTEM in the dictionary. The command

```
ADD-RELATIONSHIP
SYSTEM-CONTAINS-PROGRAM
A-SYSTEM, XY123
```

will add the entity XY123 in the dictionary as an entity of type PROGRAM and create a relationship of type SYSTEM-CONTAINS-PROGRAM with members A-SYSTEM and XY123.

2. The command

```
ADD-RELATIONSHIP
VIA RELATIONSHIP-CLASS
CONTAINS
A-SYSTEM, PROGRAM XY123
```

will have the same results as the preceding command.

3. The command

```
ADD-RELATIONSHIP
VIA RELATIONSHIP-CLASS
CONTAINS
A-SYSTEM, XY123
```

will have different results from the preceding ones. This difference comes from the fact that the entity-type of XY123 has not been specified and cannot be deduced as there exist three relationship-types in the CONTAINS relationship-class-type, i.e.

SYSTEM-CONTAINS-SYSTEM
SYSTEM-CONTAINS-PROGRAM
SYSTEM-CONTAINS-MODULE

that potentially apply to the entities specified, and consequently XY123 can either be a SYSTEM, PROGRAM or MODULE.

The execution of this command will add XY123 as an implicit entity which has a relationship of undetermined type with A-SYSTEM. A subsequent DECLARE command must then specify one of the entity-types SYSTEM, PROGRAM, MODULE, which then also results in the specification of the relationship-type an instance of which has members A-SYSTEM and XY123.

6.3.3 CHANGE-STATUS COMMAND

PURPOSE: To change the status of an entity or set of entities .

FORMAT: CHANGE-STATUS list-clause FROM status-1 TO status-2
[CHANGING VERSION [WITH DELETE]] [WITH DEPENDENCIES]

where the list-clause is one of the following:

entity-name-1 [,entity-name-2] ...
CURRENT-LIST
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-clauses

RULES:

1. There are two types of status, CONTROLLED and UNCONTROLLED. There is only one CONTROLLED status, while the dictionary administrator may specify as many uncontrolled statuses as required by the installation. Status-1 and Status-2 must be one of these installa-

tion-defined statuses, or CONTROLLED.

2. The list-clause is used to provide a list of entities that are to have their status changed either by:
 - o direct naming;
 - o using the current entity names in the qualification-list;
 - o using a set of entity names generated and stored previously in the session;
 - o using a stored procedure to generate the list;
 - o using a qualification statement.
3. Whenever an entity-name is specified in the list clause, this name must either be the primary name of an existing entity or a short-name or value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity whose status is to be changed (both prefixed by the implementor defined character \$).
4. If status-2 is the CONTROLLED status, any status-related entities must also be in CONTROLLED status prior to the "promotion" of that entity to its CONTROLLED status. Thus if A contains B and C, then A cannot be moved to CONTROLLED status unless B and C are already in CONTROLLED status. If the WITH DEPENDENCIES clause is included, then if the list-clause consists of B,A,C, then the set may all be promoted, but if this clause is not given, an error will occur, since A cannot be promoted until both B and C are in CONTROLLED status.
5. If the status of the specified entity is changed to CONTROLLED (i.e., status-2 is CONTROLLED) the same version may optionally be retained, or incremented (by using the CHANGING VERSION clause.)
6. If the status is changed from CONTROLLED (i.e., status-1 is CONTROLLED) then the version is

incremented whether the clause "CHANGING VERSION" is included or not.

7. Changing status from one uncontrolled status to another may optionally specify version change or not.

ACTIONS PERFORMED:

1. The status is changed from status-1 to status-2 for all entities identified by the list-clause.
2. Version number is increased if the change is from the CONTROLLED to any UNCONTROLLED status.
3. Version number is increased if the optional CHANGING VERSION clause is USED.
4. If the WITH DELETE clause is specified, the old version will be deleted from the dictionary.
5. If a status-related entity is in an UNCONTROLLED status any entity associated with it which is higher in the status hierarchy cannot be changed to the CONTROLLED status.
6. If the WITH DEPENDENCIES clause is used, the status of the entire set of entities is changed at the "same time", thus status-related dependencies of a hierarchical or circular variety can be resolved without resorting to any particular ordering of the entities.

ERROR CONDITIONS:

1. Status-1 or status-2 is not a known STATUS-NAME.
2. An entity named in the list-clause is not in the dictionary.
3. There is no current-list when the CURRENT-LIST option is specified.
4. The named qualification-list has not been stored

during the session.

5. The procedure-name for the RUN option does not exist in the dictionary.
6. One or more of the qualification-clauses are invalid.

EXAMPLES

1. Suppose that A CONTAINS B, C, and D, that A is in version 2, B in 2, C in 3, D in 1, and B is in CONTROLLED status, while A, C and D are not.

CHANGE-STATUS A FROM UNC-1 TO CONTROLLED

is invalid because C and D are not yet controlled.

CHANGE-STATUS A, C, D FROM UNC-1 TO CONTROLLED

is invalid. C's and D's status will be changed, but A's status was not previously changed, nor was the WITH DEPENDENCIES clause given.

At this point: B, C, D are in CONTROLLED status,
but A is not.

No version change has been made.

2. At this point A can be changed to CONTROLLED status. If we wish, the version may be incremented as follows:

CHANGE-STATUS A FROM UNC-1 TO CONTROLLED
CHANGING VERSION

It is assumed that UNC-1 is a local UNCONTROLLED option. The version of A is now 3.

3. In the above example, the old version of A (version number 2) is retained. It may be deleted after the update by:

CHANGE-STATUS A FROM UNC-1 TO CONTROLLED
CHANGING VERSION WITH DELETE

4. Now a modification is to be made to C, so it is to become UNCONTROLLED in its new version. Then the new A will become UNCONTROLLED in its new version.

CHANGE-STATUS A, C FROM CONTROLLED TO UNC-5
[CHANGING VERSION]

At this time, UNC-5 is another local UNCONTROLLED option, and whether the "changing" clause is given or not, the new versions will be made as follows:

new A version = 4
new C version = 4

5. Now we change A and C together to CONTROLLED status (presumably after other modifications) without changing their version by the command:

CHANGE-STATUS A,C FROM UNC-5 TO CONTROLLED
WITH DEPENDENCIES

6.3.4 COPY COMMAND

PURPOSE: To add an entity to the dictionary to which:

- (a) all the attributes of an existing entity are assigned, and optionally,
- (b) all the relationships, along with their attributes, of this existing entity are created.

This command is of convenience whenever a new entity is to be created that only differs in minor respects from an existing entity. The procedure that can then be followed is to edit the entity created with the COPY command, rather than to have to enter all attributes, and optionally all relationships, of the new entity.

FORMAT: Form 1 - Entity-type does not have system-generated primary names.

COPY

entity-name-1 TO entity-name-2
[WITH RELATIONSHIPS]

Form 2 - Entity-type has system-generated primary names.

COPY

entity-name-1
[WITH RELATIONSHIPS]

RULES:

1. In both Form 1 and Form 2, entity-name-1 must be the primary name or short-name or value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity to be copied (both prefixed by the implementor defined character \$).
2. Form 1 is used whenever the entity-type of entity-name-1 does not have system-generated primary names. entity-name-2 must not exist as the primary name of an entity.
3. In Form 1, entity-name-2 must be a valid primary name for the entity-type of which the entity with primary name entity-name-1 is an instance.
4. Form 2 is used whenever the entity-type of entity-name-1 has system-generated primary names.
5. In both Form 1 and Form 2, if the optional clause WITH RELATIONSHIPS is specified, the entity that is added will have the same relationships, and their associated attributes, as the entity with primary name entity-name-1.
6. The entity with primary name entity-name-1 must not be an implicit entity.

ACTIONS PERFORMED:

1. For Form 1, an entity with primary name entity-name-2 is added to the dictionary. This entity has the same attributes, and optionally, the same relationships and their attributes, as the entity with primary name entity-name-1. Both entities have the same entity-type.
2. For Form 2, an entity of the same entity-type as the entity with primary name entity-name-1 is added to the dictionary. The primary name of this entity has the same attributes, and optionally, the same relationships and their attributes, as the entity with primary name entity-name-1.
3. The command is recorded in the log/audit file.
4. Notification of the completion of the execution of the command is given to the user.
5. For the command in Form 2 the user is given the primary name of the entity that has been added to the dictionary.

ERROR CONDITIONS:

1. entity-name-1 is not the primary name or short-name or value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity to be copied (both prefixed by the implementor defined character \$).
2. entity-name-2 is specified in the case where the entity-type of the entity with primary name entity-name-1 has system-generated primary names.
3. entity-name-2 is the primary name of an entity in the dictionary.
4. entity-name-2 is not a valid primary name for the entity-type of which the entity with primary name

entity-name-1 is an instance.

5. The entity with primary name entity-name-1 is an implicit entity.

EXAMPLES

1. The command

COPY ZIP-CODE TO NEW-ZIP-CODE,

where ZIP-CODE is an ELEMENT in the dictionary, adds an entity with primary name NEW-ZIP-CODE to the dictionary. This entity has the same attributes as the entity with primary name ZIP-CODE. The command would not be allowed to execute unless in the installation standard the primary name of an ELEMENT were allowed to be 12 characters in length.

2. The command

COPY AX00127 WITH RELATIONSHIPS

adds an entity, whose primary name is generated by the system, to the dictionary. This entity will have the same attributes and relationships as the entity with primary name AX00127. The system generates a message informing the user that the entity added has the primary name AX00435, for example.

6.3.5 DECLARE COMMAND

PURPOSE: To declare the entity-type of an implicit entity.

FORMAT: DECLARE
entity-name IS entity-type-name
[clause-1 [,clause-2]...]

NOTE: In the discussion of this command it is assumed that the implicit entity was created through an ADD-RELATIONSHIP command using the VIA-RELATIONSHIP-CLASS option. Let entity-name-A denote the entity in the dictionary which was the other member of the relationship, and let entity-type-A denote the entity-type of entity-A. Let class-name denote the name of the relationship-class-type that was used in this command, and let R-name-1, ... , R-name-n denote the relationship-types that make up the relationship-group-name. For each one of these relationship-types there exists a unique entity-type which the other member of the relationship can have. Let type-1, ... , type-n denote these entity-types.

RULES:

1. entity-name must be the primary name of an implicit entity.
2. entity-type-name must be one of: type-1, ... , type-n.
3. entity-type-name must not be the name of an entity-type that is designated in the schema as having system-generated primary names.
4. entity-name must satisfy the rules for entity-names of the specified entity-type given in 6.1.1.
5. clause-1 ... are any of the attribute or attribute-group clauses of the ADD-ENTITY command.
6. The attributes and attribute-groups specified must satisfy the attribute rules given in 6.1.3.

ACTIONS PERFORMED:

1. The implicit entity with primary name entity-name becomes an instance of the entity-type declared. Let

type-j in the previous list denote this entity-type.

2. An instance of the relationship-type R-name-j with members entity-name-A and entity-name is created in the dictionary.
3. Attributes and attribute-groups as specified in clause-1 ... are assigned to the entity with primary name entity-name.
4. The command is recorded in the log/audit file.
5. Notification of the execution of the command is given to the user.

ERROR CONDITIONS:

1. entity-name is not the primary name of an implicit entity.
2. entity-type-name is not one of the names in the list: type-1, ... , type-n.
3. The entity-type specified is designated as having system-generated primary names.
4. entity-name is not a legal name for the entity-type being specified.
5. An attribute or attribute-group class is specified which does not apply to entities which are instances of the entity-type type-j.
6. entity-name is not a valid primary name for an entity of the entity-type specified.
7. An attribute or attribute-group is specified which violates the attribute rules of 6.1.3.

EXAMPLE

Suppose there exists in the dictionary schema a

relationship-class-type with name CONSISTS-OF which is composed of the following relationship-types:

BOOK-CONSISTS-OF-CHAPTERS
BOOK-CONSISTS-OF-SECTIONS
BOOK-CONSISTS-OF-PAGES

These relationship-types are between the following entity-types, respectively:

BOOK and CHAPTER
BOOK and SECTION
BOOK and PAGE

Suppose that there exists in the dictionary an entity of type BOOK with primary name FINAL-REPORT, and that the command

CREATE RELATIONSHIP
VIA RELATIONSHIP-CLASS CONSISTS-OF
BOOK, OUTLINE

has been given and that OUTLINE is not an entity in the dictionary. Execution of the command will have created an implicit entity with name OUTLINE, which can be an instance of the entity-types CHAPTER, SECTION, or PAGE.

Consider now the command

DECLARE
OUTLINE IS SECTION

The effect of this command is to make OUTLINE an instance of the entity-type SECTION, and to create an instance of the relationship-type BOOK-CONSISTS-OF-SECTIONS with members FINAL-REPORT and OUTLINE.

6.3.6 DELETE-ENTITY COMMAND

PURPOSE: To delete either a single version, or all versions of an entity in the dictionary.

FORMAT: DELETE-ENTITY
 list-name

where list-name is one of the following:

entity-name-1 [ALL VERSIONS] [,entity-name-2
 [ALL VERSIONS]] ...
CURRENT-LIST
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-clauses.

RULES:

1. entity-name-1, ... must be the primary name of an entity in the dictionary. For any such entity which is not an implicit entity, the short-name or value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity to be deleted may be used (both prefixed by the implementor defined character \$).
2. entity-name-1, ... may be the primary name of an implicit entity.
3. If entity-name-1, ... is the primary name of an implicit entity, ALL VERSIONS cannot be specified.
4. If ALL VERSIONS is not specified, the latest version of the named entity will be deleted.
5. If ALL VERSIONS is specified, all versions of the named entity will be deleted.
6. No version that is specified to be deleted can be a

member of a relationship, with exception of a relationship the other member of which is an entity of type ACCESS CONTROLLER. In the case of an implicit entity, due to the method by which this entity was added, it is not considered to be a member of a relationship.

7. The list-clause is used to provide a list of entities that are to be deleted either by:
 - o direct naming;
 - o using the entity names in the current qualification-list;
 - o using a set of entity names generated and stored previously in the session;
 - o using a stored procedure to generate the list;
 - o using a qualification command.

ACTIONS PERFORMED:

1. The version or versions specified are deleted from the dictionary.
2. For any entity which is specified to be deleted, if a relationship exists whose other member is an entity of type ACCESS CONTROLLER, this relationship is also deleted. If this ACCESS CONTROLLER entity then is not a member of any other relationship, it is also deleted. Further actions performed upon deletion of this ACCESS CONTROLLER entity are specified in Chapter 8.
3. The command is recorded in the log/audit file.
4. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. entity-name-1, ... is not the primary name of an entity in the dictionary.
2. entity-name-1, ... is the primary name of an implicit entity and ALL VERSIONS is specified.
3. A version of an entity has been specified to be deleted and this version is a member of a relationship.
4. There is no current-list when the CURRENT-LIST option is specified.
5. The named qualification-list has not been stored during the session.
6. The procedure-name for the RUN option does not exist in the dictionary.
7. One or more of the qualification-clauses are invalid.

EXAMPLES

1. DELETE-ENTITY
\$APD-X
2. DELETE-ENTITY
RUN OBSOLETE-SYSTEM-PROCEDURE

6.3.7 DELETE-RELATIONSHIP COMMAND

PURPOSE: To delete one or more relationships in a dictionary.

FORMAT: DELETE-RELATIONSHIP
{relationship-class-type-name|relationship-type-name}
list-clause
[attribute-clause]

where the list-clause is:

FROM entity-name-1 [ALL VERSIONS] TO

any one of the following:

entity-name-2 [ALL VERSIONS] [,entity-name-3
[ALL VERSIONS]] ...

CURRENT-LIST

qualification-list-name

RUN procedure-name [GIVING qualification-list-name]

QUALIFY [qualification-list-name] qualification-clauses.

where the attribute-clause is:

EITHER

attribute-type-name = attribute-1[,attribute-2]...

OR

attribute-group-type-name =
(attribute-1-1,..., attribute-n-1)
[, (attribute-1-2,..., attribute-n-2)]...

with attribute-j-1 (j=1,...m) the jth attribute of
attribute-type-name-1

RULES:

1. A relationship that is to be deleted is identified by its type-name or class-type-name, as well as the name of the two entities that it joins. If the names of the two entities are not enough for unique qualification (e.g., if there are two relationships of the same type joining these two entities (having a USAGE-NAMES attribute-group-type), and these relationships have different USAGE-NAMES attributes) the potential ambiguity may be resolved by adding an attribute-clause. In either case, all qualifying relations will be deleted.
2. Any entity-name specified must be either the primary

name of an entity or the short-name or the value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity to be modified (both prefixed by the implementor defined character \$).

3. The attribute-clause is used to identify the relationships if there are potential ambiguities.
4. Because there may be several relationships that exist because the entities have different versions, it is possible to specify that the entity named entity-name-2 or entity-name-3 is to be considered for the currently latest version, for all versions, or for a given version number. If no version clause is given, only the latest entity-name is used to qualify the relationship to be deleted; if ALL VERSIONS is specified for an entity, all specified relationships of all versions of the entity will be deleted.
5. If no relationship exists for the command, the user is informed by an error message.
6. The TO part of the list-clause may either be:
 - o given directly: entity-name-2, entity-name-3, etc.;
 - o current-list
 - o a qualification list stored previously in the session;
 - o a previously stored procedure (RUN procedure-name);
 - o a qualification command (QUALIFY ...).
7. If the GIVING clause is used in the RUN option, or if the qualification-list-name is specified in the QUALIFY option, then the resulting list is stored as the named qualification-list and the current qualification list is not changed.

8. The list-clause must have or produce relevant entity names for the named relationship-type or relationship-class-type, otherwise an error message will be generated for each invalid entity-type.

ACTIONS PERFORMED:

1. The selected relationship or set of relationships are deleted from the dictionary.
2. entity-name-1 entity-name-2, entity-name-3, etc. will be unchanged unless they are implicit (i.e., their type is unknown), in which case they are also deleted from the dictionary.
3. If any implicit entity is deleted, this fact is reported to the user.
4. If no relationship exists, the user is notified.
5. The command is recorded in the log/audit file.
6. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. The relationship-class-type-name or relationship-type-name is not valid for the dictionary.
2. The entity-names do not exist in the dictionary.
3. The entity-types of the names are not valid for the named relationship-type or relationship-class-type.
4. The relationship(s) do(es) not exist in the dictionary.
5. The version number, if given, is invalid for the target entity (entity-name-2, entity-name-3, etc.).
6. The attribute-clause contains attribute-types or

attribute-group-types invalid for the relationship.

7. There is no current-list when the CURRENT-LIST option is specified.
8. The named qualification-list has not been stored during the session.
9. The procedure-name for the RUN option does not exist in the dictionary.
10. One or more of the qualification-clauses are invalid.

EXAMPLES

1. Consider Example 1. of the ADD-RELATIONSHIP command of Section 6.3.2. The command

```
DELETE-RELATIONSHIP
  SYSTEM-CONTAINS-PROGRAM
  FROM A-SYSTEM TO XY123
```

reverses this addition.

2. Considering Examples 2. and 3. of that command, the command

```
DELETE-RELATIONSHIP
  CONTAINS
  FROM A-SYSTEM TO XY123
```

also reverses the previous command. However, if XY123 is implicit (i.e., it was not known to be of program type), then this implicit entity XY123 will also be deleted.

3. To delete all CONTAINS relationships which exist between A-SYSTEM and the latest version of XY123 with a relationship attribute-type PATH# = 2156, the command would be:

```
DELETE-RELATIONSHIP
```

CONTAINS
A-SYSTEM, XY123
PATH# = 2156

4. If the current-list contains XY123, then the same result as example 1. results from:

DELETE-RELATIONSHIP
SYSTEM-CONTAINS-PROGRAM
FROM A-SYSTEM TO CURRENT-LIST

5. If all relationships of type SYSTEM-CONTAINS-PROGRAM are to be deleted for all programs within the A-SYSTEM, and the set of these program names has previously been saved as the qualification list named PROG-IN-SYS-A, then the relationships are deleted by:

DELETE-RELATIONSHIP
SYSTEM-CONTAINS-PROGRAM
FROM A-SYSTEM TO PROG-IN-SYS-A

The qualification command would have been:

QUALIFY
SOURCE IS A-SYSTEM
FOR SYSTEM-CONTAINS-PROGRAM
GIVING TARGET

6. If the procedure that produced the above list (PROG-IN-SYS-A) was stored as PROG-SYS-A-PROC and we wish to only delete relationships that have a relationship attribute PATH# = 2156, the command could be:

DELETE-RELATIONSHIP
CONTAINS
FROM A-SYSTEM TO RUN PROG-SYS-A-PROC
PATH# = 2156

In this command, the current qualification list is replaced (because no GIVING clause was included in the RUN portion of the statement).

6.3.8 MODIFY-ENTITY COMMAND

PURPOSE: To modify attributes and/or attribute-groups of an entity existing in the dictionary.

FORMAT: MODIFY-ENTITY
 entity-name [NEW-VERSION [version-number]]
 clause-1 [,clause-2] ...

RULES:

1. entity-name must be the primary name or short-name or value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity to be modified (both prefixed by the implementor defined character \$) of an entity in the dictionary. This entity cannot be an implicit entity.
2. clause-1 ... are attribute clauses or attribute-group clauses.
3. If the optional clause NEW-VERSION is specified without version-number the next highest version number will be assigned to the entity specified and all attributes, attribute-groups, and relationships, with their associated attributes, will be COPYd to the entity with the new version number from the entity with the highest existing version number. The increment used to create the new version number from the old version number is an installation-defined number.
4. If the optional clause NEW-VERSION is specified with version-number, the number thus specified is used for the new version, all attributes, attribute-groups, and relationships, with their associated attributes, will be COPYd to the entity with the new version number from the entity with the highest existing version number. This number must be greater than the largest version number for this entity.
5. At least one attribute-clause or attribute-group

clause must be specified.

6. With exception of an attribute clause for an attribute-type whose PICTURE = TEXT, an attribute clause is of the form

attribute-type-name FROM value-1 TO value-2

to indicate the modification to take place. An attribute is deleted by specifying

FROM value-1 TO *null

and an attribute is added by specifying

FROM *null TO value-2

Whenever value-1 is not null this value must exist in the dictionary for the specified entity.

7. The form of an attribute clause for an attribute-type whose PICTURE = TEXT is given in 6.1.4.
8. An attribute-group clause is of the form

attribute-group-type-name FROM {values-1} TO
{values-2}

where {values-i} is of the form

(value-i-1, ... value-i-n)

each one of the values corresponding to an attribute-type that makes up the attribute-group-type.

An attribute-group is deleted by specifying

FROM {values-1} TO *null

and an attribute-group is added by specifying

FROM *null TO {values-2}

whenever the set {values-1} is not null this set must

exist in the dictionary for the specified entity.

9. An attribute-type whose values are system-generated cannot appear in an attribute clause.
10. Every attribute in an attribute clause or attribute-group clause is subject to the attribute rules stated in 6.1.3.
11. The security protection of an entity cannot be modified with this command.

ACTIONS PERFORMED:

1. If the optional clause NEW-VERSION is specified, the existing attributes, attribute-groups, and relationships along with their attributes, are COPYd to an entity with a new version number and all subsequent modifications are made to this entity. If, in addition, a version-number has been specified, this number is assigned to the new version.
2. The modifications specified in attribute clauses and/or attribute-group clauses are made.
3. The audit attributes are updated. If a NEW-VERSION is specified the audit attributes will be updated as if the entity being modified were a new entity in the dictionary.
4. The command is recorded in the log/audit file.
5. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. entity-name is not the primary name of an entity in the dictionary.
2. entity-name is the primary name of an implicit entity.

3. version-number is not higher than the highest existing version number for the entity specified.
4. No attribute clause or attribute-group clause is specified.
5. An attribute clause is specified which is not in the proper form.
6. An attribute-group clause is specified which is not in the proper form.
7. An attribute-type whose values are system-generated appears in an attribute clause.
8. An attribute is specified which violates the attribute rules stated in 6.1.3.
9. An attempt has been made to modify the security provisions of the entity.

EXAMPLES

1. MODIFY-ENTITY
\$PAY-NO
CLASSIFICATION FROM *null TO FINANCIAL

Here PAY-NO is the short-name of the entity PAYROLL-NUMBER.

2. MODIFY-ENTITY
XY-126
DURATION
DURATION-VALUE FROM 1 TO 20
DURATION-TYPE FROM HOUR TO MINUTES

6.3.9 MODIFY-RELATIONSHIP COMMAND

PURPOSE: To modify the attributes and/or attribute-groups of a relationship.

FORMAT: MODIFY-RELATIONSHIP
{relationship-type-name|relationship-class-type-name}
entity-name-1[ALL VERSIONS],
entity-name-2[ALL VERSIONS]
[SEQUENCE-PARAMETER = parameter]
clause-1 [,clause-2] ...

where clause-1, clause-2, ... are the clauses that specify the modifications that are to take place.

For modifications to an attribute (with exception of an attribute-type whose PICTURE = TEXT) they are of the form

attribute-type-name FROM value-1 TO value-2

to indicate the modification to take place. An attribute is deleted by specifying

FROM value-1 TO *null

and an attribute is added by specifying

FROM *null TO value-2

Whenever value-1 is not null this value must exist in the dictionary for the specified entity.

The form of an attribute clause for an attribute-type whose PICTURE = TEXT is given in 6.1.4.

For modifications to an attribute-group the clauses are of the form

attribute-group-type-name FROM {values-1}
TO {values-2}

where {values-i} is of the form

(value-i-1, ... value-i-n)

each one of the values corresponding to an attribute-type that makes up the attribute-group-type.

An attribute-group is deleted by specifying

FROM {values-1} TO *null

and an attribute-group is added by specifying

FROM *null TO {values-2}

whenever the set {values-1} is not null this set must exist in the dictionary for the specified entity.

RULES:

1. The schema must contain the relationship-type of relationship-class-type specified.
2. Entities with name entity-name-1 and entity-name-2 must exist in the dictionary.
3. The entity-type of entity-name-1 and the entity-type of entity-name-2 must be members of the relationship-type specified or a member of one of the relationship-types composing the relationship-class-type specified.
4. The relationship which is to be modified must exist in the dictionary.
5. The SEQUENCE-PARAMETER clause must apply to the relationship-type specified.
6. The attribute-types and/or attribute-group-types specified in clause-1, clause-2, ... must apply to the relationship-type an instance or instances of which is/are to be modified.

7. Each attribute and/or attribute-group which is specified in any of the clauses clause-1, clause-2, ... to be modified or deleted, must exist in the dictionary.
8. An attribute-type whose values are system-generated cannot appear in an attribute clause.

ACTIONS PERFORMED:

1. The particular relationship or set of relationships existing with members entity-name-1 and entity-name-2 are modified by their attributes being changed from the original set to the new set. Any unspecified attributes are unchanged.
2. If ALL VERSIONS is specified for an entity in the relationship, the relationships containing all versions (except for a version where the entity is in the CONTROLLED status) of that entity will be modified.
3. The command is recorded in the log/audit file.
4. Notification of the completion of the execution of the command is given to the user.

ERROR CONDITIONS:

1. The relationship-class-type-name or relationship-type-name is not valid for the dictionary.
2. An entity-name is specified which does not exist in the dictionary.
3. The entity-type of an entity-name specified is not valid for the named relationship or relationship-class.
4. The relationship specified does not exist in the dictionary.
5. The SEQUENCE-PARAMETER clause does not apply to the relationship to be modified.

6. A clause is specified which contains an attribute-type or attribute-group-type invalid for the relationship.

EXAMPLE

1. MODIFY-RELATIONSHIP
RECORD-CONTAINS-ELEMENT
PAYROLL-RECORD, EMPLOYEE-ID
VERIFICATION-DATE FROM 820115 TO 820601

This command will change the value of the attribute-type VERIFICATION-DATE (which is associated with the relationship-type RECORD-CONTAINS-ELEMENT) from 820115 to 820601 in the relationship whose members are the latest versions of the entities PAYROLL-RECORD and EMPLOYEE-ID.

2. MODIFY-RELATIONSHIP
LOCATION-HAS-DOCUMENTATION-OF-SYSTEM
AF123, PAYROLL-SYSTEM [ALL VERSIONS]
MEDIUM FROM HARD-COPY TO *null

This command deletes the attribute HARD-COPY of the attribute-type MEDIUM from the relationships of type LOCATION-HAS-DOCUMENTATION-OF-SYSTEM whose members are the location AF123 and the system PAYROLL-SYSTEM, except for the version of that system which is in the CONTROLLED status. This command would signify that all hard-copy documentation of the payroll system, except for an "operational" version had been removed from the location AF123.

6.3.10 RENAME COMMAND

PURPOSE: To change the primary name of an entity. This command applies only to an entity which is not an instance of an entity-type which has system-generated

primary names.

FORMAT: RENAME
 entity-name-1 AS entity-name-2

RULES:

1. entity-name-1 must be the primary name of an entity in the dictionary.
2. The entity-type of entity-name-1 does not have system-generated primary names.
3. There does not exist an entity with primary name entity-name-2 in the dictionary.
4. entity-name-2 must be a valid primary name for an entity of the entity-type of entity-name-1.

ACTIONS PERFORMED:

1. entity-name-1 is replaced by entity-name-2 in the entire dictionary. In particular, if entity-name-1
 - (a) has multiple versions, and/or
 - (b) exists in more than one status,the replacement extends to all such versions and/or statuses.
2. The audit attributes existing for entity-name-1 are updated and assigned to entity-name-2.
3. The entity with primary name entity-name-1 ceases to exist in the dictionary.
4. The command is recorded in the log/audit file.
5. Notification of the execution of the command is given to the user.

ERROR CONDITIONS:

1. entity-name-1 is not a primary name of an entity.
2. entity-name-1 is an instance of an entity-type which has system-generated primary names.
3. entity-name-2 is the name of an entity existing in the dictionary.
4. entity-name-2 is not a valid name for an entity of the entity-type of which entity-name-1 is an instance.

EXAMPLE

Suppose PAYROLL-FILE is an instance of the entity-type FILE in the dictionary and that it is desired to change its name to CURRENT-PAYROLL. The required command is:

RENAME PAYROLL-FILE AS CURRENT-PAYROLL

Possible error conditions encountered might be:

- (a) There currently exists an entity in the dictionary (not necessarily a file) which has the primary name CURRENT-PAYROLL.
- (b) There exists an installation standard that the primary name of an entity of type FILE must be at least 4 characters in length, but cannot exceed 14 characters. This condition would exist in the schema, where for the entity-type FILE the following meta-attribute-type specifications have been made:

MINIMUM-NAME-LENGTH = 4

MAXIMUM-NAME-LENGTH = 14

The primary name CURRENT-PAYROLL has a length

of 15 characters and hence does not represent a valid primary-name for a FILE. The RENAME command would then not be allowed to execute.

6.3.11 RENUMBER COMMAND

PURPOSE: To renumber the line numbers for attribute-types for which the value TEXT has been specified for the meta-attribute-type PICTURE.

FORMAT: RENUMBER
list-clause
{ATTRIBUTE-TYPE = attribute-type-name
FROM integer-1, integer-2
TO integer-3 [WITH-INCREMENT integer-4]}...

where list-clause is one of the following:

entity-name-1 [ALL VERSIONS] [,entity-name-2
[ALL VERSIONS]] ...
CURRENT-LIST
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-clauses.

RULES:

1. entity-name-1 ... must be the primary name of an entity or short-name or value of an attribute of an attribute type designated as USE-AS-IDENTIFIER for the entity-type of the entity whose line numbers are to be modified (both prefixed by the implementor defined character \$).
2. attribute-type-name must be the name of an attribute-type for which PICTURE = TEXT has been specified.
3. An attribute-type specified in the command must be

associated with the entity-type of the entities specified in the list-clause.

4. The list-clause is used to provide a list of entities that are to have their lines renumbered either by:
 - o direct naming;
 - o using the current entity names in the qualification-list;
 - o using a set of entity names generated and stored previously in the session;
 - o using a stored procedure to generate the list;
 - o using a qualification statement.
5. integer-1, ... , integer-4 must be positive integers with integer-1 being not greater than integer-2.
6. The default value for integer-4 is 1.

ACTIONS PERFORMED:

1. For every entity specified, the line numbers of the attributes of a designated attribute-type are changed in the following manner:
 - line (integer-1)
becomes line (integer-3)
 - line(integer-1)+1
becomes line (integer-3)+(integer-4)
 - line (integer-1)=2
becomes line (integer-3)+2x(integer-4)
 - .
 - .
 - .

- line (integer-2)
becomes line (integer-3)+
((integer-2)-(integer-1))x
(integer-4)
- 2. Any line existing in the range from (integer-3) to
(integer-3)+((integer-2)-(integer-1))x(integer-4)
which is not in the range from (integer-1) to
(integer-2) will be deleted and the line number and
contents of that line will be returned to the user.
- 3. The command is recorded in the log/audit file.
- 4. Notification of the completion of execution of the
command is given to the user.

ERROR CONDITIONS:

- 1. There is no current-list when the CURRENT-LIST option
is specified.
- 2. The named qualification-list has not been stored
during the session.
- 3. The procedure-name for the RUN option does not exist
in the dictionary.
- 4. One or more of the qualification-clauses are invalid.
- 5. An attribute-type is specified for which PICTURE =
TEXT has not been specified.
- 6. One of integer-1, ... , integer-4 has been specified
as not being a positive integer.
- 7. integer-2 has been specified as being less than
integer-1.

EXAMPLE

```
RENUMBER  
ELEMENT-LIST  
ATTRIBUTE-TYPE = DESCRIPTION  
FROM 1,1000 TO 10 WITH INCREMENT 5
```

Here it is assumed that ELEMENT-LIST is a qualification list containing the primary name of all entities of type ELEMENT in the dictionary. The command will renumber the line numbers of the attributes of type description in the following manner:

```
line 1 becomes line 10  
line 2 becomes line 15  
line 3 becomes line 20  
.  
.  
.  
line 1000 becomes line 5005
```

Any existing line in the range from 1001 to 5005 is deleted from the dictionary, and its contents and line number is returned to the user.

6.4 REPORT COMMANDS

This section contains the specification of the commands that are available for producing reports from the dictionary. The following commands (listed in alphabetic order by command name) are being specified:

CATALOG
IMPACT-OF-CHANGE
IMPLICIT-ENTITIES
LIST
ORPHANED-ENTITIES
PRODUCE-SYNTAX
VERSION-REPORT

6.4.1 CATALOG COMMAND

PURPOSE: To produce a report on selected entities that shows their attributes and attribute-groups, as well as specified relationships together with their attributes and attribute-groups. An option exists to show only specified attribute-type values in order to provide the user with a degree of customization.

FORMAT: CATALOG
 Report-Set
 Relationship-Set
 attribute-type-specification-clause
 Report-sequence
 [REPORT-TITLE [EVERY-PAGE] = "string"]
 [Destination]

where Report-Set is one or more of the following:

ALL
entity-type-name-1 [,entity-type-name-2]...
entity-name-1 [,entity-name-2]...
CURRENT-LIST
qualification-list-name

RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-
clauses

where Relationship-Set is one of the following:

ALL-RELATIONSHIPS

The name(s) of one or more relationship-type(s)

The name(s) of one or more relationship-class-
type(s)

The name(s) of one or more relationship-type-
chain(s)

where attribute-type-specification-clause is

[attribute-type-name-1 [,attribute-type-name-2] ...]
[attribute-group-type-name-1 [attribute-group-type-
name-2] ...]|ALL-ATTRIBUTES

where, if the relationship-type-chain option is not
used, the Report-Sequence clause is one of the
following:

- a) SEQUENCE: REPORT-SET ENTITY-NAME
REPORT-SET ENTITY-TYPE-NAME
RELATIONSHIP-TYPE-NAME
RELATED ENTITY-NAME
- b) SEQUENCE: REPORT-SET ENTITY-TYPE-NAME
REPORT-SET ENTITY-NAME
RELATIONSHIP-TYPE-NAME
RELATED ENTITY-NAME

where, if the relationship-type-chain option is used,
the Report-Sequence clause is one of the following:

- a) SEQUENCE: REPORT-SET ENTITY-NAME
- b) SEQUENCE: REPORT-SET ENTITY-TYPE-NAME
REPORT-SET ENTITY-NAME

and where the Destination clause is one of the

following:

- a) device-name
- b) name of desired location of output

RULES:

1. The Report-Set clause selects the specific qualification list that is to be used as the target set for the report. At least one of the alternatives must be given. If more than one is given, the resulting list will be their union.
2. The current-list will only be changed if the RUN option is used with the GIVING clause, or if the QUALIFY option is used with the qualification-list-name option included.
3. The use of ALL provided a catalog of all entities existing in the dictionary.
4. If ALL-RELATIONSHIPS is specified in the Relationship-Set, all relationships in which an entity of above is a member will also be reported on. For each such relationship, the attributes and attribute-groups of the relationship, as well as the name and entity-type of the other member of the relationship will be reported. Implicit entities will also be reported with the relationship-class-type that was specified in their creation.
5. The name of a relationship-type in the Relationship-Set must be such that it applies to at least one entity-type of the entities that are being reported on.
6. The name of a relationship-class-type in the Relationship-Set must be such that it applies to at least one entity-type of the entities that are being reported on.

7. A relationship-type-chain is a sequence

relationship-type-1, ... , relationship-type-n

where consecutive relationships in the sequence have a common entity-type, and where loops are not permitted. All entities that are members of a relationship whose relationship-type is a component of a chain will be reported on, and all attributes and attribute-groups of the relationship-types will be shown. relationship-type-1 must have as a member an entity-type which pertains to the entities being reported on.

8. Entities that are members of relationships of the relationship-types that are components of a relationship-type-chain will be reported in the order of the chain.
9. Every entity-type-name must exist in the dictionary schema.
10. Unless a specific version-number is specified for a Report-Set entity, the entity with the highest version number will be reported on. Related entities reported on will be those with the highest version number occurring in the relationship.
11. The optional REPORT-TITLE clause specifies the title that is to be shown on the report. If the EVERY-PAGE subclause is given, this title will appear on every page of the report. The maximum length of the string that can be specified is implementor-defined.
12. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. For every entity to be reported on, the following will be produced:
 - a) The primary name of the entity and its entity-type.

- b) The attributes and attribute-groups corresponding to the attribute-types and attribute-group-types that have been specified.
2. For every Related entity that is being reported, the primary name of the entity and its entity-type will be shown.
 3. If an entity-name is specified which does not exist in the dictionary, the execution of the command will not abort, but a warning message will be issued.
 4. If a destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. An entity-type-name which does not exist in the schema is specified.
2. An entity-name which does not exist in the dictionary is specified.
3. There does not exist a current-list, i.e. the user has not issued a QUALIFY command in the current session.
4. A qualification-list-name which does not exist is specified.
5. The specified procedure-name is not known to the system.
6. The qualification-clauses refer to a non-existent entity-type-name and/or entity-name.
7. The relationship-type-chain option has not been used and a relationship-type has been specified that does not pertain to the entities being reported on.
8. The relationship-type-chain option is being used and a chain has been specified which does not meet the

conditions of Rule 7.

9. An invalid attribute-type-name and/or attribute-group-type name is specified.
10. The location specified in the Destination clause is not a valid location.

EXAMPLES

1. CATALOG
FILE, RECORD, ELEMENT
CONTAINS
SEQUENCE: REPORT-SET ENTITY-TYPE-NAME
REPORT-SET ENTITY-NAME
RELATIONSHIP-TYPE-NAME
RELATED ENTITY-NAME

This command produces the following report:

- a. the word FILE .
- b. the primary name of entities of type FILE, in alphabetic order, with their attributes.
- c. for each such entity, the relationship-type-name of the relationship-class-type (in this case FILE-CONTAINS-RECORD), followed by its attributes and/or attribute groups, and then by the primary names of all entities of type RECORD that participate in a relationship with the entity of type FILE.
- d. the word RECORD
- e. the primary name of entities of type RECORD, in alphabetic order, with their attributes.
- f. the word RECORD-CONTAINS-ELEMENT followed by the attributes and/or attribute groups of the relationship, and then the primary names of all entities of type ELEMENT that participate in a relationship with an entity of type

RECORD appearing in e. above.

2. CATALOG
ALL
SEQUENCE: ENTITY-NAME

This command produces a report showing all entities in the dictionary. The entities will be ordered alphabetically by primary name, and all their attributes and attribute-groups will be given. Only the latest version will be shown unless the user has indicated that entities in the CONTROLLED status are to be reported on.

3. CATALOG
FILE-A, FILE-B
SEQUENCE: ENTITY-NAME

This command produces a report giving FILE-A and FILE-B, with all their attributes and attribute-groups.

4. CATALOG
CURRENT-LIST
SEQUENCE: ENTITY-TYPE-NAME, ENTITY-NAME
PRESIDENTS-PRINTER

This command produces a report of all entities in the current qualification list with all their attributes and attribute groups. The output order is all entities of a given type printed in sequence (i.e., the major key is entity-type-name, and the minor key is entity-name). The report is sent to the "President's Printer".

5. CATALOG
QUALIFY ELEMENT LANGUAGE = COBOL
SEQUENCE: ENTITY-NAME

This command produces a report of all elements that

have a COBOL name, gives all their attributes and attribute-groups, with the order being alphabetical by primary name of the entities.

6. CATALOG
RUN PROCEDURE-QL-1
LENGTH, DESCRIPTION
SEQUENCE: ENTITY-NAME

It is assumed that there exists a stored procedure with name PROCEDURE-QL-1 that generates entities of type RECORD; the command will execute this procedure and will produce a report of all these entities showing their type (i.e. RECORD) along with their LENGTH attribute and DESCRIPTION. The order will be alphabetic by primary name of the entities.

7. CATALOG
ZIPLIST
DESCRIPTION, CLASSIFICATION
SEQUENCE: ENTITY-NAME

This report contains the entities with their primary name and entity-type which are contained in the qualification list ZIPLIST. Also shown are the attributes of type DESCRIPTION and CLASSIFICATION of these entities. The order is alphabetic by primary name of the entities.

6.4.2 IMPACT-OF-CHANGE COMMAND

PURPOSE: To provide a capability to identify all entities in the dictionary which might be impacted in some manner by a change to designated entities. The resulting report is similar to the LIST command report, except that the contents of this report are further restricted by the semantics of the command to include only potentially impacted entities.

FORMAT: IMPACT-OF-CHANGE

[Report-Set]
[Relationship-Set]
Report-Sequence
[REPORT-TITLE [EVERY-PAGE] = "string"]
[Destination]

where Report-Set is one of the following:

ALL
entity-type-name-1 [,entity-type-name-2]...
entity-name-1 [,entity-name-2]...
CURRENT-LIST
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-
clauses

where Relationship-Set is one of the following:

ALL-RELATIONSHIPS
ALL-RELATIONSHIP-CHAINS
The name(s) of one or more relationship-type(s)
The name(s) of one or more relationship-class-
type(s)
The name(s) of one or more relationship-type-
chain(s)

Whenever the "one or more relationship-type-chain"
option is not used, the Report-Sequence clause will be
one of the following:

- a) SEQUENCE: REPORT-SET PRIMARY-NAME
REPORT-SET ENTITY-TYPE-NAME
RELATIONSHIP-TYPE-NAME
RELATED PRIMARY-NAME
RELATIONSHIP-CLASS-TYPE-NAME
RELATED IMPLICIT-ENTITY-NAME
- b) SEQUENCE: REPORT-SET ENTITY-TYPE-NAME
REPORT-SET PRIMARY-NAME
RELATIONSHIP-TYPE-NAME

RELATED PRIMARY-NAME
RELATIONSHIP-CLASS-TYPE-NAME
RELATED IMPLICIT-ENTITY-NAME

where the expressions serve to distinguish between entities that exist in the (optionally qualified) Report-Set and those that are members of the relationships being reported on.

When the relationship-type-chain option is used the Report-Sequence clause is one of the following:

- a) SEQUENCE: REPORT-SET ENTITY-NAME
- b) SEQUENCE: REPORT-SET ENTITY-TYPE-NAME
REPORT-SET ENTITY-NAME

Entities that are members of relationships of the relationship-types that are components of a relationship-type-chain will be reported in the order of the chain.

and where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. The Report-Set clause selects the specific qualification list that is to be used as the target set for the report. At least one of the alternatives must be given. If more than one is given, the resulting list will be their union.
2. The current-list will only be changed if the RUN option is used with the GIVING clause, or if the QUALIFY option is used with the qualification-list-name option included.
3. The use of ALL provided a catalog of all entities

existing in the dictionary.

4. If ALL-RELATIONSHIPS is specified in the Relationship-Set, all relationships in which the entity (to be reported on) is a member will also be reported on. For each such relationship, the name, usage-name and the entity-type of the member of the relationship will be reported. Implicit entities and the relationship-class-type that was used in their creation will also be reported.
5. If ALL-RELATIONSHIP-CHAINS is specified in the Relationship-Set, all chains are "traced" to identify all entities which are either directly related (because a relationship exists in the dictionary) or derivable by tracing the relationships, without looping, to identify all entities which are related by implication to the source entity. The resulting entities and relationships will be reported on per the ALL relationship described above.

A relationship-type-chain is a sequence

relationship-type-1, ... , relationship-type-n

where consecutive relationships in the sequence have a common entity-type, and where loops are not permitted. All entities that are members of a relationship whose relationship-type is a component of a chain will be reported on, and all attributes and attribute-groups of the relationship-types will be shown. relationship-type-1 must have as a member an entity-type which pertains to the entities being reported on.

6. If a relationship-type or relationship-class-type name is specified in the Relationship-Set, the relationships and entities are restricted to the types specified. The report elements are the same as those mentioned above.
7. The name of a relationship-type in the Relationship-Set must be such that it applies to at least one entity-type of the entities that are being reported on.

8. The name of a relationship-class-type in the Relationship-Set must be such that it applies to at least one entity-type of the entities that are being reported on.
9. Unless a specific version-number is specified for a Report-Set entity, impact-of-change will be determined for the entity with the highest version-number.
10. All versions of related entities will be reported on. The sequence of reporting will be in descending version-number sequence within relationship-type-name.
11. The optional REPORT-TITLE clause specifies the title that is to be shown on the report. If the EVERY-PAGE subclause is given, this title will appear on every page of the report. The maximum length of the string that can be specified is implementor-defined.
12. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. For every "impacted" entity, the following is produced:
 - a) The primary-name of the entity and the entity's type.
 - b) The names of all relationship-types or relationship-class-types it is involved in with respect to the source entity and as qualified by the relationship-set.
2. If an entity-name is specified which does not exist in the dictionary, the execution of the command will not abort, but a warning message will be issued.
3. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. An entity-type-name is specified which does not exist in the schema.
2. An entity-name which does not exist in the dictionary is specified.
3. There is no current-list, i.e. the user has not issued a QUALIFY command in the current session.
4. A qualification-list-name which does not exist is specified.
5. The specified procedure-name is not known to the system.
6. The qualification-clauses refer to a non-existent entity-type-name and/or entity-name.
7. The relationship-type-chain option has not been used and a relationship-type has been specified that does not pertain to the entities being reported on.
8. The relationship-type-chain option is being used and a chain has been specified which does not meet the conditions of Rule 5.
9. The location specified in the Destination clause is not a valid location.

EXAMPLES

1. IMPACT-OF-CHANGE
RUN FILE-PROC
ALL-RELATIONSHIPS
SEQUENCE: REPORT-SET PRIMARY NAME
REPORT-SET ENTITY-TYPE-NAME
RELATIONSHIP-TYPE-NAME
RELATED PRIMARY-NAME
RELATIONSHIP-CLASS-TYPE-NAME

RELATED IMPLICIT-ENTITY-NAME
PRESIDENTS-PRINTER

The primary names of entities that qualify by running the procedure are examined to determine which entities they are related to (through any possible relationship). The output is then sequenced as follows:

- a) The primary names of the original entities that qualified;
- b) Their entity-types;
- c) The primary names of the entities to which they are related;
- d) The name of the relationship-class-type that links them;
- e) The names of any implicit entities.

2. IMPACT-OF-CHANGE

QUALIFY ENTITY-TYPE ELEMENT CONTAINS

SEQUENCE: REPORT-SET ENTITY-TYPE-NAME
REPORT-SET PRIMARY-NAME
RELATIONSHIP-TYPE-NAME
RELATED PRIMARY-NAME
RELATIONSHIP-CLASS-TYPE-NAME
RELATED IMPLICIT-ENTITY-NAME

Here a list of all primary names of elements is first produced and then the relationship class CONTAINS is used to determine all other entities (of several types, such as FILE and RECORD) that are related to the elements by this relationship class. A report is then generated with a slightly different sequence to the ones in the first example.

3. IMPACT-OF-CHANGE

QUALIFY ENTITY ZIP FOR ENTITY-TYPE ELEMENT ONLY
FILE-CONTAINS-ELEMENT

REPORT-DERIVED-FROM-FILE
FILE-PROCESSED-BY-SYSTEM
SEQUENCE: REPORT-SET-ENTITY-NAME
PRESIDENTS-PRINTER

All elements which have an alternate name of ZIP are selected. For each of these elements which is part of a FILE-CONTAINS-ELEMENT relationship, the files are "selected". For each of these files which is in either a REPORT-DERIVED-FROM-FILE or FILE-PROCESSED-BY-SYSTEM relationship, the corresponding REPORT and/or system entity is "selected". The resulting report presents the selected elements in alphabetical sequence. For each one of these elements, the selected relationships and their types/attributes are presented. The results are sent to the president's printer.

6.4.3 IMPLICIT-ENTITIES COMMAND

PURPOSE: To generate a report on the implicit entities in the dictionary along with the entities that were used in their creation.

FORMAT: IMPLICIT-ENTITIES
 [REPORT-TITLE [EVERY-PAGE] = "string"]
 [Destination]

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. The optional REPORT-TITLE clause specifies the title that is to be shown on the report. If the EVERY-PAGE subclause is given, this title will appear on every

page of the report. The maximum length of the string that can be specified is implementor-defined.

2. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. A report is generated in alphabetic sequence that shows for each implicit entity in the dictionary
 - a) the name of the implicit entity, and its audit attributes,
 - b) the name of the relationship-class-type that was used in its creation,
 - c) the name of the entity that was used in its creation, along with all of its attributes and attribute-groups.
2. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. The location specified in the Destination clause is not a valid location.

EXAMPLE

IMPLICIT-ENTITIES
PRESIDENTS PRINTER

6.4.4 LIST COMMAND

PURPOSE: To produce a report which contains the primary names of entities of designated entity-types.

FORMAT: LIST
 {ALL|entity-type-name-1 [,entity-type-name-2]
 ... }
 [REPORT-TITLE [EVERY-PAGE] = "string"]
 [Destination]

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. Every entity-type-name specified must be the name of an entity-type in the schema.
2. The optional REPORT-TITLE clause specifies the title that is to be shown on the report. If the EVERY-PAGE subclause is given, this title will appear on every page of the report. The maximum length of the string that can be specified is implementor-defined.
3. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. If ALL is specified, the report will include all entity-types.
2. For each entity-type specified, the report will list (all versions) of all entities of that entity-type, showing (in alphabetic sequence by entity-type-name and entity-name)

- a) the primary name of the entity and its entity-type,
 - b) the attributes of attribute-type STAGE and STATUS,
 - c) the audit attributes.
3. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. An invalid entity-type-name has been specified.
2. The location specified in the Destination clause is not a valid location.

6.4.4A ORPHANED-ENTITIES COMMAND

PURPOSE: To generate a report on the orphaned entities in the dictionary, i.e., entities that are not members of any relationship.

FORMAT: ORPHANED-ENTITIES
 [REPORT-TITLE [EVERY-PAGE] = "string"]
 [Destination]

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. The optional REPORT-TITLE clause specifies the title that is to be shown on the report. If the EVERY-PAGE subclause is given, this title will appear on every page of the report. The maximum length of the string that can be specified is implementor-defined.
2. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. A list of all orphaned entities is generated in alphabetic sequence by entity-type and within entity-type by primary name in alphabetic sequence.
2. For each entity listed, all attributes and attribute-groups of the entity will be presented in a manner similar to the CATALOG command.
3. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. The location specified in the Destination clause is not a valid location.

EXAMPLE

```
ORPHANED-ENTITIES
REPORT-TITLE EVERY PAGE = "ORPHANED ENTITIES REPORT -
JANUARY 1, 1983"
DICT-ADM-PRINTER
```


6.4.4B PRODUCE-SYNTAX COMMAND

PURPOSE: To generate a report, for a selected set of entities, of all attributes, attribute-groups, and relationships of these entities in a format of ADD-ENTITY and ADD-RELATIONSHIP commands.

FORMAT: PRODUCE-SYNTAX
 Report Set
 [Destination]

where Report-Set is one or more of the following:

ALL
entity-type-name-1 [,entity-type-name-2]...
entity-name-1 [,entity-name-2]...
CURRENT-LIST
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-
 clauses

and where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. The Report-Set clause selects the specific qualification list that is to be used as the target set for the report. At least one of the alternatives must be given. If more than one is given, the resulting list will be their union.
2. The current-list will only be changed if the RUN option is used with the GIVING clause, or if the QUALIFY option is used with the qualification-list-

name option included.

3. The use of ALL provided a catalog of all entities existing in the dictionary.
4. Every entity-type-name must exist in the dictionary schema.
5. Unless a specific version-number is specified for a Report-Set entity, the entity with the highest version number will be reported on. Related entities reported on will be those with the highest version number occurring in the relationship.
6. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. For each entity in the Report Set, the report will list:
 - the primary name of the entity and its entity-type
 - the attributes and attribute-groups of the entity in the format of how they would appear in an ADD-ENTITY command.
 - for every relationship in which this entity is a member, the name of a relationship-type of which this relationship is an instance and the primary name of the entity which is the other member of the relationship.
 - the attributes and attribute-groups of this relationship in the format of how they would appear in an ADD-RELATIONSHIP command.
2. The order of this report is the following:
 - alphabetic by primary name of entity in the Report Set;

- alphabetic by name of the relationship-type;
 - alphabetic by primary name of the entity which is the other member of the relationship;
 - attribute clauses are in alphabetic order of the attribute-type name;
 - attribute-group clauses are in alphabetic order of the attribute-group-type name.
3. The default for the Destination clause is an implementor option.

ERROR CONDITIONS:

1. An entity-type-name which does not exist in the schema is specified.
2. An entity-name which does not exist in the dictionary is specified.
3. There does not exist a current-list, i.e. the user has not issued a QUALIFY command in the current session.
4. A qualification-list-name which does not exist is specified.
5. The specified procedure-name is not known to the system.
6. The qualification-clauses refer to a non-existent entity-type-name and/or entity-name.
7. The location specified in the Destination clause is not a valid location.

6.4.5 VERSION-REPORT COMMAND

PURPOSE: To generate a report on all versions of selected entities.

FORMAT: VERSION-REPORT
 Report-Set
 Report-sequence
 [REPORT-TITLE [EVERY-PAGE] = "string"]
 [Destination]

where Report-Set is one of the following:

ALL
entity-type-name-1 [,entity-type-name-2]...
entity-name-1 [,entity-name-2]...
CURRENT-LIST
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]



QUALIFY [qualification-list-name] qualification-
clauses

where the Destination clause is one of the following:

- a) device-name
- b) name of desired location of output

RULES:

1. The Report-Set clause selects the specific qualification list that is to be used as the target set for the report. At least one of the alternatives must be given. If more than one is given, the resulting list will be their union.
2. The current-list will only be changed if the RUN option is used with the GIVING clause, or if the QUALIFY option is used with the qualification-list-name option included.
3. The use of ALL provided a catalog of all entities existing in the dictionary.
4. The optional REPORT-TITLE clause specifies the title that is to be shown on the report. If the EVERY-PAGE subclause is given, this title will appear on every page of the report. The maximum length of the string that can be specified is implementor-defined.
5. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. For each Report-Set entity version the following will be reported:
 - a) All attributes and attribute-groups of the version
 - b) All relationships in which the version parti-

cipates, along with the attributes and attribute-groups of the relationship and the primary name of the other member of the relationship (if more than one version of that entity exists in such relationships, only the one with the highest version number will appear).

- c) The names of the implicit entities that were created using the Report-Set entity as the other member in the ADD-RELATIONSHIP command.
2. If an entity-name is specified which does not exist in the dictionary, the execution of the command will not abort, but a warning message will be issued.
 3. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.

ERROR CONDITIONS:

1. An entity-type-name is specified which does not exist in the schema.
2. A qualification-list-name is specified which does not exist.
3. There does not exist a current-list, i.e. the user has not issued a QUALIFY command in the current session.
4. An invalid qualifying clause has been specified.
5. The location specified in the Destination clause is not a valid location.

6.5 QUERY COMMANDS

While the reporting capabilities can be used to generate a short report, which may be returned on-line to the user terminal, the use of a report capability is generally less user friendly than using a query facility. This section discusses formats of simple queries that can be used to generate simple outputs. In general they use the qualification clauses of section 6.2.1 as basic building blocks, but the query uses special keywords to simplify the user's task.

In generating a set of useful commands for on-line queries, it was assumed that the full set of qualification clauses should be available for the general query statement, but that these could be replaced by a qualification list. The command structure is in two parts: The first part deals with a generalized query command and operations on that command; and the second part is composed of a set of specialized queries.

Every query command will first return a count of qualified entities to the user; at this point the user has the option of directing further output from the query to an alternate location with a Destination clause.

6.5.1 GENERAL QUERY COMMAND

PURPOSE: A general query command has almost the same power as a general reporting command except that the format of the output is not specified by the user and the output is not expected to be large (though this is not a restriction), and hence headers and page counts are not required.

FORMAT: QUERY [query-name]

{qualification-list-name|CURRENT-LIST|WORK
 procedure-name|QUALIFY qualify-clause}

[GIVING {ALTERNATE-NAME [AND CONTEXT]|START ENTITY|
 attribute-type [,attribute-type]...}...]|ALL
 ATTRIBUTE-TYPES

RULES:

1. The Query may optionally be named. If a name is given, it may be reused through a SAVE-QUERY command (as discussed later).
2. The qualification list may be selected through providing the name of an available list (i.e., one that was previously constructed), or by requesting the current-list, or by generating it (RUN) through a previously stored procedure (procedure-name), or by providing a QUALIFY statement.
3. The current-list will not be changed as the result of applying the Query command.
4. Output, with no GIVING subclause, is the list of entity names that were in the qualification list.
5. The use of the alternate name provides 'pairs' of the entity-name (primary) and any alternate name(s). If the context clause is added, the alternate name context is also included.
6. If the RUN or QUALIFY options are used, then the alternate name context, if used, in the alternate-name-selection-clause (see 6.2.1.2) will select relevant contexts (e.g., if this clause is for COBOL, then any PL/I contexts will be ignored).
7. The use of the START ENTITY clause will mean special print out of the entities that were selected using a relationship-restriction-clause. The special print-out will show which starting entity-name is related to

which other entity name and its relationship-type.

8. Any specified attribute-types will be output along with the entity-type for which they are relevant; irrelevant attribute-types will be ignored, but if no entity-type is appropriate to the attribute-types specified, a warning message is sent to the user.
9. The use of the creation and update subclause will cause all such relevant data to be output with each entity.

ERROR CONDITIONS:

1. Any error in a RUN or QUALIFY command will cause an abort of the command. If the qualification-list-name is not known to the system, the command aborts. In each case, the reason for aborting is made known to the user.
2. An attribute-type exists which does not pertain to an entity of the qualification-list.

6.5.2 SAVE-QUERY COMMAND

PURPOSE: To save a query previously named and used in the session.

FORMAT: SAVE-QUERY query-name
[SAVE-TEXT (text)]

RULES:

1. The query-name must be a previously named query, generated during the session.
2. Optional text may be added to produce a non-executed comment, available to the user for explanation.

3. Any saved text may be renewed by GIVE-~~SAVED~~-TEXT.

ACTIONS PERFORMED:

1. The query that was previously named is stored (in some implementor defined location) with any optional text.

ERROR CONDITIONS:

1. The query-name is unknown to the system. The command aborts and the user is informed.

6.5.3 RUN-QUERY COMMAND

PURPOSE: To execute a previously saved query.

FORMAT: RUN-QUERY query-name

RULES:

1. The query-name must refer to a previously stored command. If the system does not have any knowledge of the named query, the command aborts with a comment to the user.
2. The "text", if any exists, is ignored in the execution of the query. (See 6.5.2)

ERROR CONDITIONS:

1. The query-name does not refer to a previously stored query.

6.5.4 DELETE-QUERY COMMAND

PURPOSE: To delete a previously stored query.

FORMAT: DELETE-QUERY query-name

RULES:

1. The query name must refer to a previously stored command. The system will comment if it cannot find the referenced query and the command will abort.
2. The user will be provided positive response upon execution of the query.

ERROR CONDITIONS:

1. The query-name does not refer to a previously stored query.

6.5.5 LIST-QUERIES COMMAND

PURPOSE: To give a list of all stored queries.

FORMAT: LIST-QUERIES
[TEXT ALSO]
[PROCEDURE ALSO]

RULES:

1. A count is appended to the output list.
2. Optionally any text can also be printed.

ACTIONS PERFORMED:

1. A list of all stored queries is output.
2. Any text is given with the name, if requested.
3. The original stored material is then optionally given.

ERRORS:

1. None.

6.5.6 SPECIAL QUERY COMMANDS

There are several special queries that can be called for easier user selection. They are each separately discussed in this section.

6.5.6.1 IMPLICIT-ENTITY-QUERY

PURPOSE: To output a list of entities that were defined via a relationship-class-type and have not since been DECLARED by the user.

FORMAT: QUERY [qualification-list-name] IMPLICIT

RULES:

1. The qualification-list-name will be the means of later referring to this list. If it is absent, the result will be in the current-list. The current-list is unchanged if a qualification-list-name is given or if the command is aborted.
2. A count of qualified entities is output prior to the

implicit list.

3. The user will have the option to prevent output from being returned to the terminal.

ERROR CONDITIONS:

1. None.

6.5.6.2 ENTITY-AND-ATTRIBUTE QUERY

PURPOSE: To output a list of primary-names and optionally, to include the attributes of specified attribute-types and/or the attribute-groups of specified attribute-group-types.

FORMAT: QUERY [qualification-list-name]

[[ENTITY-TYPE entity-type-name-1 [,entity-type-name-2]...]

[ATTRIBUTE-TYPE attribute-type-name-1[,attribute-type-name-2]...]

[ATTRIBUTE-GROUP-TYPE attribute-group-type-name-1[,attribute-group-type-name-2]...]...]

[[ENTITY entity-name-1 [,entity-name-2]...]

[ATTRIBUTE-TYPE attribute-type-name-1[,attribute-type-name-2]...]

[ATTRIBUTE-GROUP-TYPE attribute-group-type-name-1[,attribute-group-type-name-2]...]...]

RULES:

1. The qualification-list-name will be the means of later referring to this list. If it is absent, the result will be in the current-list. The current-list is unchanged if there is a name or if the command is aborted.

2. The attribute-type-name(s) and/or attribute-group-type-name(s), if included, must be valid for the corresponding entity-type-name, or the user receives an error message and the query is aborted.
3. At least one ENTITY-TYPE or ENTITY subclause must be given.
4. A count of qualified entities is returned to the user before results are output. The user will have the option to prevent output from coming back to the terminal.

ERROR CONDITIONS:

1. An attribute-type-name or attribute-group-type-name is not applicable to the entity-type-name or the entity-name, respectively.
2. There is no ENTITY-TYPE or ENTITY subclause.

6.5.6.4 ALTERNATE-NAME-AND-CONTEXT QUERY

PURPOSE: To select all relevant entities in a given context or to produce name-context pairs for a given primary name.

FORMAT: QUERY [qualification-list-name]

{ENTITY entity-name-1[,entity-name-2]...|ENTITY-TYPE entity-type-name}

{CONTEXT = alternate-name-context|NAME AND CONTEXT}

RULES:

1. The qualification-list-name will be the means of later

referring to this list. If it is absent, the result will be in the current-list. The current-list is unchanged if there is a name or if the command is aborted.

2. The entity list may contain a number of entity-names joined with commas. Only one entity-type is allowed per query.
3. Either the context may be specified, or the set of alternate name/context pairs will be supplied for all qualifying entities.
4. A count of qualified entities is returned before the results are output. The user will have the option to prevent output from being returned to the terminal.

ERROR CONDITIONS:

1. An invalid entity-type-name is specified.

6.5.6.4 CONTAINS-STRING QUERY

PURPOSE: To return the names of entities that contain a given string in their primary name, alternate-name, description, or comments.

FORMAT: QUERY [qualification-list-name]

STRING "string (with * for don't-care)"

IN{PRIMARY-NAME|ALTERNATE-NAME|
DESCRIPTION|COMMENTS|ALL}

RULES:

1. The qualification-list-name will be the means of later referring to this list. If it is absent, the result

will be a new current-list. The current-list is unchanged if there is a name or if the command is aborted.

2. The string may contain * to show places that any character may exist and the string will match (e.g., "B*G" matches BUG, BIG, DEBUG, CRIBBAGE, etc.).
3. The output consists of the primary-names of entities that qualify and the relevant values of alternate-name and context, description, or comments attribute-type.
4. A count of qualified entities is returned before the results are output. The user will have the option to prevent output from being returned to the terminal.

ERROR CONDITIONS:

1. None.

6.5.6.5 RELATIONSHIP QUERY

PURPOSE: To allow queries based on a given relationship for a certain entity or for a given entity-type.

FORMAT: QUERY [qualification-list-name]

RELATIONSHIP FOR {STARTING ENTITY entity-name
|ENTITY-TYPE entity-type-name}

IS {relationship-class-type-name|relationship-
type-name}

RULES:

1. The qualification-list-name will be the means of later referring to this list. If it is absent, the result will be a new current-list. The current-list is

unchanged if there is a name or if the command is aborted.

2. For the entity-type-name option, only a relationship-type-name must be given, and it must be relevant to the entity-type specified.
3. The starting entity-name option requires a relationship-class-type-name; or a relevant relationship-type name.
4. A count of qualified entities is returned before the results are output. The user will have the option to prevent output from being returned to the terminal.

ERROR CONDITIONS:

1. The entity-name or entity-type-name must be meaningful to the relationship-class-type-name or the relationship-type-name.

6.5.6.6 AUDIT QUERY

PURPOSE: To list all primary-names of entities with a certain audit condition(s).

FORMAT: QUERY [qualification-list-name]

ENTITIES {CREATED BEFORE date|CREATED SINCE date|
CREATED BY person-name|LAST-CHANGED BEFORE date|
LAST-CHANGED SINCE date|LAST-CHANGED BY person-
name}...

RULES:

1. The qualification-list-name will be the means of later referring to this list. If it is absent, the result will be in the current-list. The current-list is

unchanged if there is a name or if the command is aborted.

2. The dates must be valid or the person-names must be valid. The use of more than one subclause allows further restriction (by ANDing the results).
3. A count of entities that qualify is returned to the user. The user will have the option to prevent output from being returned to the terminal.

ERROR CONDITIONS:

1. A person-name is specified which is not valid.

6.5.6.7 CLASSIFICATION QUERY

PURPOSE: To list all primary-names of entities with certain CLASSIFICATION keywords.

FORMAT: QUERY [qualification-list-name]

CLASSIFICATION [ENTITY-TYPE entity-type-name]

KEYWORD = keyword-1 [{AND|OR} keyword-2]

RULES:

1. entity-type-name must be the name of an existing entity-type.
2. The number of entities that qualify are returned to the user prior to the entity-names being listed.
3. A count of qualified entities is returned before the results are output. The user will have the option to prevent output from being returned to the terminal.

ERROR CONDITIONS:

1. entity-type-name is not the name of an existing entity-type.



CHAPTER 7. DDS SOFTWARE INTERFACES

This chapter is composed of three major sections.

- o Section 7.1 specifies the facilities that are available to produce from the dictionary programming language representations of the data used within a COBOL program.
- o Section 7.2 specifies the commands that are available for the EXPORT/IMPORT facility of the DDS, through the use of which it is possible to transfer a selected portion of the contents of one dictionary of a standard DDS to another dictionary of a standard DDS.
- o Section 7.3 specifies a facility whereby the DDS may be called from a program.

7.1 GENERATE-STRUCTURE-FOR-COBOL COMMAND

PURPOSE: To produce a programming language representation of the data used within a COBOL program. The DATA DIVISION, including the File and Working-Storage Sections, can be produced using this command. All ANS COBOL rules must apply.

FORMAT: GENERATE-STRUCTURE-FOR-COBOL
 entity-name {FILE-SECTION|Working-Storage-
 Section-Clause}

[Using-Clause]
[Uniqueness-Clause]
[Prefix-Suffix-Clause]
[Destination]

where entity-name is the primary name of a FILE or
RECORD entity

where Working-Storage-Section-clause is

WORKING-STORAGE-SECTION

{DATA|CONDITION|REDEFINES}

where Using-Clause is

USING {{USAGE-NAME|PRIMARY-NAME|ALTERNATE-NAME}
 AND {ENTITY-CHARACTERISTICS|
 RELATIONSHIP-CHARACTERISTICS}}

where Uniqueness-Clause is

UNIQUENESS {REQUIRED|NOT REQUIRED}

where prefix-suffix clause is

{PREFIX text-string|SUFFIX text-string}

and where DESTINATION Clause is one of the following:

- a) device name
- b) name of desired location of output. The new
 default is implementor defined.

RULES:

1. If FILE-SECTION is specified a COBOL file description is produced for the specified file entity. The file description will be created using:
 - a) For the COBOL FD clauses, the USAGE-FORMAT attribute in the REPRESENTATION Attribute-Group of the file is used.
 - b) For the COBOL record description(s), the FILE-CONTAINS-RECORD relationship-type is used to determine the records associated with the file. The COBOL record description (01 level) clauses for each relationship are obtained from the text of either the relationship's or the "contained" record's REPRESENTATION Attribute-Group as determined by the USING-clause. If a record entity is specified, the latter case occurs.
 - c) To determine the data in the record description, the RECORD-CONTAINS-ELEMENT relationship is used.
 - (1) The characteristics of the COBOL elementary items are determined from either the relationship's or the contained element's REPRESENTATION Attribute-Group as determined by the USING-clause.
 - (2) To determine those elements which define a group within the record description, the contained elements are examined to determine if any is a source entity in an ELEMENT-CONTAINS-ELEMENT relationship. For each entity that satisfies this condition, if the USAGE-INDICATOR attribute in that relationship is GROUP, then a group structure is generated with the containing elements (which may be groups).
 - (3) To determine the relative position of the element in the record, the attribute REL-POSITION is used. If a gap is left between elements, a FILLER element with appropriate

picture length is placed in the generated record description.

- (4) A special case may occur where a user may want to use already documented RECORD entities to "create" the definition of another record. This can be accomplished by using the RECORD-CONTAINS-RECORD relationship. If this happens, the contained records are treated as groups and thus have a level in the record description which is greater than the source record of the relationship.

- d. The record description name used for each file, record, group or element identified above is determined by the USING-clause.

2. If WORKING-STORAGE-SECTION and DATA are specified:

- a. For each RECORD entity designated or each one contained in the designated file, a 01-level description and associated elements/group are generated as in 1a - 1d above.
- b. For each ELEMENT entity involved in a FILE-CONTAINS-ELEMENT relationship with the specified file, a 77-level elementary item is produced.
- c. The name and characteristics used in each case are determined by the USING-clause.

3. If WORKING-STORAGE-SECTION and CONDITION are specified:

- a. For each RECORD entity specified or for each one contained in the designated file, a 01-level description is produced as in 1a above. For all ELEMENT entities related to a selected RECORD entity by a RECORD-CONTAINS-ELEMENT relationship, an 88-level elementary item is generated, if the USAGE-INDICATOR in the relationship is CONDITION.
- a. The name and characteristic used in each case is determined by the USING-clause.

4. If WORKING-STORAGE-SECTION and REDEFINES are specified:
 - a. For each RECORD entity designated or each one contained in the designated file, if the RECORD entity is a source entity for a RECORD-CONTAINS-RECORD relationship and the USAGE-INDICATOR in the relationship is REDEFINES, the contained records are assumed to be redefinitions of the source entity.
 - b. Similarly, if in an ELEMENT-CONTAINS-ELEMENT relationship, the USAGE-INDICATOR is REDEFINES, the contained element(s) is assumed to be a redefinition of the source element.
 - c. The rules mentioned above apply to the generation of the appropriate record, group and element descriptions.
5. The default for the Using-Clause is ALTERNATE-NAME AND RELATIONSHIP-CHARACTERISTICS.
6. If UNIQUENESS is REQUIRED, the alternate name of the record associated with COBOL is appended to the name of each elementary item or group in the specified record description of the file section. UNIQUENESS applies only to the file section.
7. The default for the Uniqueness-Clause is REQUIRED.
8. If FILE-SECTION is specified, entity-name must be the primary name of an entity of type FILE.
9. If WORKING-STORAGE-SECTION is specified, entity-name must be the primary name of an entity of either type RECORD or ELEMENT.
10. The length of the name selected by the USING-clause or its default cannot exceed the allowed length of an ANS COBOL data-name. The selected name will be truncated from the right to make it the maximum length allowed, if it is greater.

11. If the Prefix-Suffix-Clause is used, the text-string is appended to the beginning or end of the data-name in the structure generated. Rule 10 applies here.
12. The default for the Destination clause is an implementor option.

ACTIONS PERFORMED:

1. For the designated FILE entity, a COBOL FD with associated record, group and element descriptions is generated, if the FILE-SECTION clause is used.
2. If the WORKING-STORAGE-SECTION clause is used, all record, group and element descriptions are generated to meet specified criteria.
3. If PRIMARY-NAME is designated, but the primary name is the name of an entity whose type is not allowed for the conditions specified, the command will abort, and a warning message will be issued.
4. If a Destination clause is given, the report will either be output on the device specified or "placed" in the named location.
5. A notification of completion of execution will be provided to the user.
6. If an invalid USING or UNIQUENESS clause is specified, the clause is ignored and defaults are assumed.

ERROR CONDITIONS:

1. An entity of the wrong type is specified for generation.
2. An entity of the wrong type is specified for the clause used in the command.
3. An invalid FILE-SECTION clause is specified.

4. An invalid WORKING-STORAGE-SECTION clause is specified.
5. The attributes which should correspond to the designated clauses, do not.
6. A REL-POSITION attribute causes an overlap of elements. A warning message is given, but the command does not abort.

EXAMPLES

1. GENERATE-STRUCTURE-FOR-COBOL
PAYROLL
FILE-SECTION
USING ALTERNATE-NAME AND ENTITY-CHARACTERISTICS

Assuming that PAYROLL is an entity of entity-type file, a COBOL FD is generated. Any information which appears in the USAGE-FORMAT of PAYROLL is made part of the PAYROLL FD paragraph following ANSI COBOL rules. For records involved in a FILE-CONTAINS-RECORD relationship with PAYROLL, a sequence of 01 level paragraphs is created. For each element involved in a RECORD-CONTAINS-ELEMENT relationship with the records mentioned above, an elementary item or group definition is created. The information to be associated with each of these records, groups or elements is determined by the USAGE-FORMAT attribute of each. Assume that the following description is generated.

FD PAYROLL BLOCK CONTAINS 10 RECORDS
LABEL RECORD IS STANDARD

01 PAYREC.

02 DATE.

03 DAY PIC 99.
03 MONTH PIC A(9).
03 YEAR PICTURE 9(4).

- (a) The "BLOCK CONTAINS 10 RECORDS" clause must be in the USAGE-FORMAT of the PAYROLL file entity.
- (b) The LABEL RECORD IS STANDARD is not required to be in the USAGE-FORMAT, because it is the COBOL default.
- (c) Nothing appeared in the USAGE-FORMAT of the PAYREC record entity.
- (D) The PIC 99, PIC A(9), and PICTURE 9(4) all came from the USAGE-FORMATS of the DAY, MONTH, YEAR elements.
- (e) For each of the names PAYROLL, PAYREC, DATE, DAY, MONTH and YEAR the alternate name associated with COBOL was used.

2. GENERATE-STRUCTURE-FOR-COBOL

PAYREC

WORKING-STORAGE-SECTION REDEFINES

USING ALTERNATE-NAME AND ENTITY-CHARACTERISTICS

As for example 1., an 01 level description of PAYREC is generated. The difference in this case is that either a RECORD-CONTAINS-RECORD relationship exists and contains a USAGE-INDICATOR with value REDEFINES, or the REDEFINES exists in a RECORD-CONTAINS-ELEMENT or ELEMENT-CONTAINS-ELEMENT relationship.

Assume that the following description is generated:

01 PAYREC.

02 DATE.

03 DAY PIC 99.

03 MONTH PIC A(9)

03 YEAR PICTURE 9(4).

02 DATE-COMPOSITE REDEFINES DATE PIC X(15).

01 PAY-KEY REDEFINES PAYREC.

02 KEY PIC X(6).

02 FILLER PIC X(9).

In this case, there was both an ELEMENT-CONTAINS-ELEMENT and RECORD-CONTAINS-RECORD relationship with USAGE-INDICATOR equal to REDEFINES. In these cases, DATE was related to DATE-COMPOSITE and PAYREC was related to PAY-KEY. In both cases the reserved word REDEFINES and the name of the entity being redefined were automatically placed in the structure, based on the relationships.

7.2 THE EXPORT/IMPORT FACILITY

This section specifies the commands that are available in the DDS core standard that permit a selected part of one dictionary to be transferred to another dictionary, where it is naturally assumed that both dictionaries are part of a standard DDS. It is however **not** assumed that both dictionaries are managed by the same implementation of the core standard Dictionary Processing System. Through the use of this facility it is then possible to transfer, for example, a portion (or all) of a dictionary named DICTIONARY-1 (which is a part of implementation-A of the standard DDS) to a dictionary named DICTIONARY-2 (which belongs to implementation-B of the standard DDS).

7.2.1 INTEGRITY CONSIDERATIONS

In order for this facility to be practical and reliable a number of different factors must be considered:

- o Since both implementations of the core standard of the DDS have extensibility facilities, it is possible that the two schemas involved may have been customized

differently. This not only means that DICTIONARY-1 may contain entities, relationships, attributes, etc., such that the type of these does not exist in the schema of DICTIONARY-2, but also that the integrity constraints specified in the two schemas may be different. Examples of such possible discrepancies are:

The name of elements in DICTIONARY-1 can be 24 characters long, but are limited to 18 characters in DICTIONARY-2.

The list of legal attributes for an attribute-type may be different in the two dictionaries.

The number of allowable attributes for entities of a given type may be different.

- o A resolution must exist as to how to deal with entities which are being transferred, where entities by that name already exist in the target dictionary. In this respect consideration must also be given to version numbers of entities in DICTIONARY-1 and the manner in which they are to be dealt with in DICTIONARY-2.
- o Similar problems are possible in the differences in STATUS-NAME meta-entities, and a reconciliation with existing entities is required.

The facility specified here is designed to assure that exercise of the EXPORT/IMPORT commands will preserve the integrity of the target dictionary. This will be achieved, along with other precautionary restrictions, by requiring that these commands will only be permitted between dictionaries whose "essential schemas" are identical. In the following sections there will be given the definition of this concept, along with the specification of commands required for its operation on schemas.

7.2.2 SCHEMA EQUIVALENCE AND THE ESSENTIAL SCHEMA

Given a schema of a dictionary the essential schema of this dictionary is defined to be composed of the following:

1. The set of names of all the meta-entities in the schema.
2. The set of meta-relationships in the schema.
3. The meta-attributes of the following types:

For an entity-type meta-entity:

MINIMUM-NAME-LENGTH

MAXIMUM-NAME-LENGTH

PICTURE

SYSTEM-GENERATED

For a relationship-type meta-entity:

SEQUENCED

SEQUENCE-PARAMETER

For a relationship-class-type meta-entity:

none

For an attribute-type meta-entity:

MINIMUM-LENGTH

MAXIMUM-LENGTH

For an **attribute-group-type** meta-entity:

none

For an **attribute-type-validation-procedure** meta-entity:

none

For an **attribute-type-validation-data** meta-entity:

VALUE/RANGE

DATA-VALUE

DATA/RANGE

For a **status-name** meta-entity:

none

For a **stage-name** meta-entity:

none

For a meta-relationship of type **M-R-T(relationship-type, entity-type)**:

POSITION

For a meta-relationship of type **M-R-T(relationship-type, attribute-type)**:

SINGULAR/PLURAL

MAXIMUM-NUMBER-OF-OCCURRENCES

For a meta-relationship of type **M-R-T(relationship-class-type, relationship-type)**:

none

For a meta-relationship of type M-R-T(attribute-group-type, attribute-type):

GROUP-POSITION

For a meta-relationship of type M-R-T(entity-type, attribute-group-type):

SINGULAR/PLURAL

MAXIMUM-NUMBER-OF-OCCURRENCES

For a meta-relationship of type M-R-T(relationship-type, attribute-group-type):

SINGULAR/PLURAL

MAXIMUM-NUMBER-OF-OCCURRENCES

For a meta-relationship of type M-R-T(attribute-type, attribute-type-validation-procedure):

none

For a meta-relationship of type M-R-T(attribute-type, attribute-type-validation-data):

none

Based on this definition of essential schema, two schemas are said to be **essentially equivalent** if their essential schemas are identical.

7.2.2.1 EXTRACT-ESSENTIAL-SCHEMA COMMAND

PURPOSE: To generate the essential schema of a dictionary in machine-readable form.

FORMAT: EXTRACT-ESSENTIAL-SCHEMA
 dictionary-name
 location-clause

where location-clause is the name of a file or an
implementor defined type of location.

RULES:

1. dictionary-name must be the name of an existing dictionary.
2. location-clause must specify a valid location.
3. Additional clauses may be required by an implementor.

ACTIONS PERFORMED:

1. The essential schema of the named dictionary is written to the location specified in the location-clause in the SCHEMA-EXPORT-FORMAT.
2. The SCHEMA-EXPORT-FORMAT consists of:
 - a) The name of the dictionary.
 - b) The names of the meta-entities in the essential schema along with their type, sequenced by meta-entity-type-name and meta-entity-name, each meta-entity being followed by its meta-attributes sequenced by meta-attribute-type-name and meta-attribute.
 - c) The meta-relationships in the essential schema along with their type, sequenced by meta-entity-type-name and the name of the first meta-entity in the meta-relationship, each meta-relationship being followed by the meta-attributes of the meta-relationship sequenced by meta-attribute-type-name and meta-attribute.

3. The user is informed of the completion of the execution of the command.

ERROR CONDITIONS:

1. dictionary-name is not the name of an existing dictionary.
2. The location specified in the location-clause is not a valid location in the system.

7.2.2.2 COMPARE-ESSENTIAL-SCHEMAS COMMAND

PURPOSE: To compare two essential schemas existing in SCHEMA-EXPORT-FORMAT and to inform the user if either they are identical or what differences exist between them.

FORMAT: COMPARE-ESSENTIAL-SCHEMAS
dictionary-name-1 AT location-1 AND dictionary-name-2
AT location-2

RULES:

1. The essential schema of the dictionary with name dictionary-name-1 must exist in SCHEMA-EXPORT-FORMAT at location-1.
2. The essential schema of the dictionary with name dictionary-name-2 must exist in SCHEMA-EXPORT-FORMAT at location-2.

ACTIONS PERFORMED:

1. The two essential schemas are compared, and:
 - a) If they are identical, the user is informed of this fact.

- b) If they are not identical, the user is informed of the differences that exist between them, e.g., a type exists in schema-1 but not in schema-2.

ERROR CONDITIONS:

1. location-1 does not contain the essential schema of dictionary-name-1 in SCHEMA-EXPORT-FORMAT.
2. location-2 does not contain the essential schema of dictionary-name-2 in SCHEMA-EXPORT-FORMAT.
3. dictionary-name-1 and/or dictionary-name-2 are not valid names for dictionaries.

7.2.3 EXPORT/IMPORT PROCEDURE

Suppose there exists a dictionary with name SOURCE-DICTIONARY, and that it is desired to extract a subset of SOURCE-DICTIONARY and to merge this subset into another existing dictionary named TARGET-DICTIONARY. This subset is defined by means of the qualification commands specified in Section 6.2 and the same type of specification of relationship-types which is used in the CATALOG command of Section 6.4.1.

The procedure to be followed consists of the following steps:

1. An EXTRACT-SUBSET command is issued to operate on SOURCE-DICTIONARY. This command:
 - a) Extracts the desired subset from SOURCE-DICTIONARY in DICTIONARY-EXPORT-FORMAT, and
 - b) Invokes the EXTRACT-ESSENTIAL-SCHEMA

command to operate on SOURCE-DICTIONARY,
and

- c) Places the results in a designated location.
2. A dictionary with name TEMPORARY-DICTIONARY is created with the command CREATE-DICTIONARY.
 3. The results of Step 1. are placed in TEMPORARY-DICTIONARY with the command LOAD-DICTIONARY.
 4. The contents of TEMPORARY-DICTIONARY is operated on using the maintenance commands of Section 6.3 to achieve the following:
 - a) The dictionary contains no more than one version of any entity.
 - b) All entities are in a single status.
 5. The schema of TEMPORARY-DICTIONARY is operated on using the schema maintenance commands of Section 5.1 in order to make it equivalent to the schema of TARGET-DICTIONARY. Deletion of unwanted descriptors may require compensating changes to the contents of TEMPORARY-DICTIONARY. The EXTRACT-ESSENTIAL-SCHEMA command can be used on TEMPORARY-DICTIONARY and TARGET-DICTIONARY, and the results evaluated with the COMPARE-ESSENTIAL-SCHEMAS command to ensure that the required equivalence has been achieved.
 6. Once the required equivalence has been achieved, Step 1. is repeated, this time operating on TEMPORARY-DICTIONARY.
 7. At this point the desired objective can be achieved by the command IMPORT-SUBSET issued to operate on TARGET-DICTIONARY and specifying the location designated for the output of Step 6. Options will have to be specified on the actions to be taken on the status-name in the schema of TARGET-DICTIONARY to be used, as well as the version numbers of

entities to be assigned for entities that are being imported where entities with the same primary names already exist in TARGET-DICTIONARY.

Special consideration must be given to entity-types for which it has been specified that primary names are to be system-generated. If such entities are included in the dictionary descriptors which are the subject of the EXPORT/IMPORT process no assurance exists that these primary names are synchronized in the two dictionaries being dealt with. Moreover, if such is the case, neither DDS can have knowledge of what is required to establish such a reconciliation. If a reconciliation is required, it will have to be handled as a human process, and the changes required should be made on the descriptors when they are in the DICTIONARY-EXPORT-FORMAT at the completion of either Step 1. or Step 6. above. Tools required to make such changes are an implementor option. The same consideration applies to attribute-types whose values have been specified as being system-generated.

In the discussion of this section it has been assumed that both SOURCE-DICTIONARY and TARGET-DICTIONARY reside on the same computer system. Should this not be the case, the output of Step 1. will have to be transported from the computer system on which SOURCE-DICTIONARY resides to the computer system on which TARGET-DICTIONARY resides. Tools required to achieve this are outside the scope of this specification.

7.2.4 EXPORT/IMPORT COMMANDS

In this section the commands required by the preceding procedure will be specified. The following general observations should be noted:

- o All commands, with the exception of the CREATE-DICTIONARY command, operate on a single dictionary, and thus are issued as operating on that dictionary. This means that the required authority for the command can be stored as a part of that dictionary. Details on this point will be given in Chapter 8.

- o The CREATE-DICTIONARY command is issued at the DDS level, and may in fact have to be issued before there is a single dictionary in existence. Authority to issue this command will have to be established as a part of the log-on process.

7.2.4.1 EXTRACT-SUBSET COMMAND

PURPOSE: a) To extract a specified subset from an existing dictionary and make it available in a standard format for further processing.

 b) To invoke at the same time the EXTRACT-ESSENTIAL-SCHEMA command.

FORMAT: EXTRACT-SUBSET
 DICTIONARY dictionary-name
 TO extract-name
 Extract-Set
 Relationship-Set
 attribute-type-specification-clause
 location-clause

where Extract-Set is one or more of the following:

ALL
entity-type-name-1 [,entity-type-name-2]...
entity-name-1 [,entity-name-2]...
qualification-list-name
RUN procedure-name [GIVING qualification-list-name]
QUALIFY [qualification-list-name] qualification-
 clauses

where Relationship-Set is one of the following:

ALL-RELATIONSHIPS
The name(s) of one or more relationship-type(s)
The name(s) of one or more relationship-class-
 type(s)

The name(s) of one or more relationship-type-chain(s)

where attribute-type-specification-clause is

```
[attribute-type-name-1 [,attribute-type-name-2] ...]  
[attribute-group-type-name-1 [attribute-group-type-  
name-2] ... ]|ALL-ATTRIBUTES
```

RULES:

1. dictionary-name must be the name of a dictionary.
2. The Extract-Set clause selects the specific qualification list that is to be used as the target set for the command. At least one of the alternatives must be given. If more than one is given, the resulting list will be their union.
3. The use of ALL provides a subset consisting of all entities in the dictionary.
4. If ALL-RELATIONSHIPS is specified in the Relationship-Set, all relationships in which an entity of above is a member will also be extracted on. For each such relationship, the attributes and attribute-groups of the relationship, as well as the name and entity-type of the other member of the relationship will be extracted. Implicit entities will also be extracted with the relationship-class-type that was specified in their creation.
5. The name of a relationship-type in the Relationship-Set must be such that it applies to at least one entity-type of the entities that are being reported on.
6. The name of a relationship-class-type in the Relationship-Set must be such that it applies to at least one entity-type of the entities that are being extracted.

7. A relationship-type-chain is a sequence

relationship-type-1, ... , relationship-type-n

where consecutive relationships in the sequence have a common entity-type, and where loops are not permitted. All entities that are members of a relationship whose relationship-type is a component of a chain will be extracted along with all attributes and attribute-groups of the relationship-types. relationship-type-1 must have as a member an entity-type which pertains to the entities being extracted.

8. Every entity-type-name, relationship-type-name, relationship-class-type-name, attribute-type-name, and attribute-group-type-name specified must exist in the schema of the named dictionary.
9. Unless a specific version-number is specified for an Extract-Set entity, the entity with the highest version number will be extracted. Related entities extracted will be those with the highest version number occurring in the relationship.
10. The location-clause must specify a valid location.
11. The `DICTIONARY-EXPORT-FORMAT` consists of the following:
- a) The names of the entities along with their type, sequenced by entity-type and primary name of the entity, each entity being followed by:

the attributes of the entity and their types, sequenced by attribute-type-name and attribute

the attribute-groups of the entity and their types, sequenced by attribute-group-type-name and attribute-group.

Implicit entities will be denoted by the literal "IMPLICIT" in place of the entity-type.

- b) The relationships along with their type, sequenced

by relationship-type-name and the name of the first entity in the relationship, each relationship being followed by:

the attributes of the relationship and their types, sequenced by attribute-type-name and attribute

the attribute-groups of the relationship and their types, sequenced by attribute-group-type-name and attribute-group.

For implicit entities, the relationship-class-type-name will be used in place of the relationship-type-name.

ACTIONS PERFORMED:

1. The essential schema of the named dictionary is placed at the designated location in SCHEMA-EXPORT-FORMAT.
2. The specified subset of the named dictionary is placed at the designated location in DICTIONARY-EXPORT-FORMAT.
3. The user is informed of the completion of execution of the command.

ERROR CONDITIONS:

1. dictionary-name is not the name of a dictionary.
2. An entity-type-name which does not exist in the schema is specified.
3. An entity-name which does not exist in the dictionary is specified.
4. A qualification-list-name which does not exist is specified.
5. The specified procedure-name is not known to the

system.

6. The qualification-clauses refer to a non-existent entity-type-name and/or entity-name.
7. The relationship-type-chain option has not been used and a relationship-type has been specified that does not pertain to the entities being reported on.
8. The relationship-type-chain option is being used and a chain has been specified which does not meet the conditions of Rule 7.
9. An invalid attribute-type-name and/or attribute-group-type name is specified.
10. The location specified in the Destination clause is not a valid location.

7.2.4.2 CREATE-DICTIONARY COMMAND

PURPOSE: To create a dictionary with a specified name. The schema of this dictionary can be specified as being either the standard schema, an existing schema, or the essential schema of an existing schema.

FORMAT: CREATE-DICTIONARY
DICTIONARY-NAME IS dictionary-name-1
[SCHEMA IS dictionary-name-2 [ESSENTIAL] SCHEMA]

RULES:

1. dictionary-name-1 must not be the name of an existing dictionary.
2. The default to the optional SCHEMA IS clause is the system-standard schema.

3. dictionary-name-2 must be the name of an existing dictionary.
4. Additional clauses may be required by the implementor.

ACTIONS PERFORMED:

1. The named dictionary is created. If no SCHEMA IS clause has been specified, this dictionary will have the system-standard schema.
2. If a SCHEMA IS clause has been specified, the dictionary will have the schema (or optionally the essential schema) of the dictionary dictionary-name-2.
3. The dictionary created will be "empty", i.e., this term denoting a dictionary that does not have any user-supplied dictionary descriptors. An "empty" dictionary contains a single entity of type DICTIONARY-USER, with an implementor defined dictionary user name. This dictionary user is assigned the full set of permissions to the entire functionality of the DDS. The name of this dictionary user can subsequently be changed through the use of the RENAME command of Section 6.3.10.
4. The audit-meta-attributes in the schema will show the date of creation and identification of the person issuing the command.
5. The action will be recorded in the log/audit file of the dictionary dictionary-name-1.
6. The user is informed of the completion of the execution of the command.

ERRORS:

1. There exists a dictionary with name dictionary-name-1.
2. There does not exist a dictionary with name dictionary-name-2.

7.2.4.3 LOAD-DICTIONARY COMMAND

PURPOSE: a) To provide an existing essential schema in SCHEMA-EXPORT-FORMAT to an existing "empty" dictionary.

 b) To load an existing dictionary subset in DICTIONARY-EXPORT-FORMAT into an "empty" dictionary.

FORMAT: LOAD-DICTIONARY
 DICTIONARY dictionary-name
 WITH extract-name
 AT location-name

RULES:

1. dictionary-name must be the name of an "empty" dictionary with system-standard-schema.
2. extract-name must specify the result of an EXTRACT-SUBSET command.
3. extract-name must reside at the location specified by location-name.
4. Additional clauses may be required by an implementor.

ACTIONS PERFORMED:

1. The named dictionary is assigned the essential schema contained in extract-name.
2. The dictionary-descriptors in extract-name are loaded into the named dictionary.
3. The audit-meta-attribute-types of the schema are updated.

4. The audit-attribute-types of the dictionary are updated.
5. The command is recorded in the log/audit file.
6. The user is informed of the completion of the execution of the command.

ERRORS:

1. dictionary-name is not the name of an "empty" dictionary with system-standard schema.
2. extract-name does not reside at the specified location.

7.2.4.4 IMPORT-SUBSET COMMAND

PURPOSE: To merge descriptors previously extracted from a dictionary into another dictionary.

FORMAT: IMPORT-SUBSET
INTO DICTIONARY dictionary-name
FROM extract-name AT LOCATION location-name
TARGET-STATUS IS status-name
VERSION-OPTION IS {NEW|LATEST}
IMPLICIT-ENTITY-OPTION IS {REPLACE|IGNORE}

RULES:

1. dictionary-name must be the name of an existing dictionary.
2. extract-name must be the result of an EXTRACT-SUBSET command residing at location-name.
3. The essential schema which is part of extract-name

must be identical to the essential schema of the dictionary with name dictionary-name.

4. The dictionary descriptors in extract-name cannot specify entities of type `DICTIONARY-USER` or `ACCESS-CONTROLLER`.
5. status-name must be the name of a status-name in the schema of dictionary-name.
6. status-name cannot be `SECURITY-STATUS`, as discussed in Chapter 8.
7. status-name cannot be the `CONTROLLED` status.
8. If the `VERSION-OPTION` specified is `NEW`, then, in the case where there exists an entity in the designated status in dictionary-name which has the same primary name as an entity in extract-name, an entity with the next highest version number will be added.
9. If the `VERSION-OPTION` specified is `LATEST`, then, in the case where there exists an entity in the designated status in dictionary-name which has the same primary name as an entity in extract-name, the entity in extract-name will replace the existing entity with the highest version number.
10. If the `IMPLICIT-ENTITY-OPTION` specified is `NEW`, then, in the case where there exist implicit entities with the same primary name in extract-name and dictionary-name, the implicit entity in dictionary-name will be deleted and replaced by the implicit entity in extract-name.
11. If the `IMPLICIT-ENTITY-OPTION` specified is `IGNORE`, then, in the case where there exist implicit entities with the same primary name in extract-name and dictionary-name, the implicit entity in extract-name will be ignored.
12. Additional clauses may be required by an implementor.

ACTIONS PERFORMED:

1. The dictionary descriptors in extract-name are merged into the designated status in dictionary-name in accordance with the designated VERSION-OPTION and IMPLICIT-ENTITY-OPTION options.
2. The audit-attribute-types in the dictionary are updated.
3. The command is recorded in the log/audit file.
4. The user is informed of the completion of the execution of the command.

ERROR CONDITIONS:

1. There does not exist a dictionary with name dictionary-name.
2. extract-name does not exist at the specified location.
3. There exists a dictionary descriptor in extract-name which specifies an entity of type DICTIONARY-USER or ACCESS-CONTROLLER.
4. The essential schema of dictionary-name and the essential schema in extract-name are not identical.
5. status-name is not the name of a status-name in the schema.
6. SECURITY-STATUS has been specified in the TARGET-STATUS clause.
7. The meta-attribute of type CONTROLLED/UNCONTROLLED of status-name is CONTROLLED.
8. A required clause has been omitted.

7.3 DDS PROGRAM INTERFACE - THE CALL DDS COMMAND

PURPOSE: To provide access to the DDS from a program written in a standard language which has a CALL facility.

FORMAT: CALL DDS
{log-on}
command-1 [,command-2] ...

RULES:

1. The log-on must specify the name of the dictionary or dictionaries that will be accessed by the command or commands.
2. The command(s) command-1 ... must be identical to the manner in which they are submitted to the DDS by a user of the DDS.
3. Only the following command names may appear in a command:

EXTRACT-ESSENTIAL-SCHEMA
COMPARE-ESSENTIAL-SCHEMAS
EXTRACT-SUBSET
LOAD-DICTIONARY
IMPORT-SUBSET

4. Additional clauses may be required by an implementor.

ACTIONS PERFORMED:

1. The commands are executed in the sequence in which they are given.
2. Acknowledgement of the completion of execution of the commands specified is given to the program that issued

the CALL DDS command.

3. If an error is detected in the execution of one of the specified commands, execution of that command is aborted, and control is returned to the program that issued the CALL DDS command, along with the appropriate error message.
4. Each command is recorded in the appropriate log/audit file.
5. Upon completion of the execution of all of the given commands, control is returned to the program that issued the CALL DDS command.

ERROR CONDITIONS:

1. The error conditions for each one of the individual commands apply.

CHAPTER 8. DICTIONARY ADMINISTRATOR COMMANDS AND TOOLS

This chapter contains the specification of the security facility of the core standard DDS and the description of tools for the Dictionary Administrator that are to be made available by an implementor of the core standard DDS.

8.1 THE DDS SECURITY FACILITY

The DDS security facility consists of three levels of access control:

- o The first level controls the access to the DDS itself and a specified dictionary. This level of control is provided by the implementor of the DDS in a manner which is an option of the implementor. It is assumed that the identity of a dictionary user who has been validated at this level will be passed to the DDS in the form of a dictionary-user-name.
- o The second level of control, which will be called the **global** level, occurs through dictionary entities of type **DICTIONARY-USER** and their attributes. These entities specify the permissions that have been granted to a dictionary user in terms of the commands that are allowed to execute against specific entity-types in designated statuses. Attributes can also be specified to indicate that stated relationship-types are not included in the view which a user has of the dictionary. Other attributes of these entities are also used to specify privileges of a dictionary user with respect to the schema of the dictionary.
- o The third level of control, which will be called the **local** level, occurs through dictionary entities of type **ACCESS-CONTROLLER** and their attributes. Any

dictionary entity can be protected at this local level by establishing a relationship in the dictionary between it and such an entity of type ACCESS-CONTROLLER. As part of the creation of an entity of this type in the dictionary, the DDS also creates an attribute of type READ-LOCK and an attribute of type WRITE-LOCK. Attributes of type READ-KEY and WRITE-KEY can be assigned to designated dictionary users and in a subsequent section of this chapter there will be specified what keys are required for the execution of specific commands on an entity which has local protection. Global protection has precedence over local protection in the sense that local protection will not be checked unless the dictionary user has global permission for the command on the entity-type and status of the entity in question.

8.1.1 DICTIONARY-USER ENTITY-TYPE

Entities of type **DICTIONARY-USER** serve to provide the specification of global permissions for users of a dictionary system. The definition of this entity-type given in Chapter 4. is:

ENTITY-TYPE NAME = **DICTIONARY-USER**

PURPOSE = "To describe individuals who are users of the data dictionary system and to record their access privileges to the dictionary. Entities of this type are used exclusively in the management of the security facility of the dictionary system, and are not available through its generally available facilities."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard

schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.

LAST-MODIFIED-IN-SCHEMA-BY

will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

CONNECTABLE = NO

ENTITY-CLASS = SECURITY

The following **attribute-group-type** is associated with this entity-type:

ATTRIBUTE-GROUP-TYPE NAME = **DICTIONARY-PERMISSIONS**

PURPOSE = "To specify the access permissions to the dictionary that have been granted to a dictionary user."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

which is composed of the following attribute-types (in the order in which they are in the attribute-group-type):

1. ATTRIBUTE-TYPE NAME = **STATUS**

(as previously defined)

2. ATTRIBUTE-TYPE NAME = ENTITY-TYPE-NAME

PURPOSE = "To specify the entity-type name or group of entity-type names for which permissions are being declared. Legal attributes of this type are:

the name of an entity-type in the dictionary schema

ALL

where ALL denotes all entity-types in the schema."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

3. ATTRIBUTE-TYPE NAME = ADD-PERMISSION

PURPOSE = "The ADD-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the following commands:

ADD-ENTITY

ADD-RELATIONSHIP

COPY

DECLARE

RENAME

All Qualification Commands of Section 6.2

All Report Commands of Section 6.4

All Query Commands of Section 6.5

The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

4. ATTRIBUTE-TYPE NAME = **DELETE-PERMISSION**

PURPOSE = "The DELETE-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the following commands:

DELETE-ENTITY

DELETE-RELATIONSHIP

All Qualification Commands of Section 6.2

All Report Commands of Section 6.4

All Query Commands of Section 6.5

The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

5. ATTRIBUTE-TYPE NAME = **MODIFY-PERMISSION**

PURPOSE = "The MODIFY-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the following commands:

MODIFY-ENTITY

MODIFY-RELATIONSHIP

RENUMBER

All Qualification Commands of Section 6.2

All Report Commands of Section 6.4

All Query Commands of Section 6.5

The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

6. ATTRIBUTE-TYPE NAME = READ-PERMISSION

PURPOSE = "The READ-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the following commands:

All Qualification Commands of Section 6.2

All Report Commands of Section 6.4

All Query Commands of Section 6.5

The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

7. ATTRIBUTE-TYPE NAME = STATUS-PERMISSION

PURPOSE = "The STATUS-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the following commands:

CHANGE-STATUS

All Qualification Commands of Section 6.2

All Report Commands of Section 6.4

All Query Commands of Section 6.5

The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

8. ATTRIBUTE-TYPE NAME = LOAD-PERMISSION

PURPOSE = "The LOAD-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the LOAD-DICTIONARY command. The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

9. ATTRIBUTE-TYPE NAME = **IMPORT-PERMISSION**

PURPOSE = "The IMPORT-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the IMPORT-SUBSET command. The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

10. ATTRIBUTE-TYPE NAME = **PROTECT-PERMISSION**

PURPOSE = "The PROTECT-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of the following commands, described in Section 8.2:

ADD-SECURITY
DELETE-SECURITY
MODIFY-SECURITY
ASSIGN-KEY
DELETE-KEY

The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

11. ATTRIBUTE-TYPE NAME = **ADMINISTRATOR-PERMISSION**

PURPOSE = "The ADMINISTRATOR-PERMISSION attribute-type, if given the value YES, permits the designated dictionary user the use of all the commands of the DDS. It is the permission required to add, modify, or delete

entities of type **DICTIONARY-USER**. The default value is **NO**."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

This entity-type is also associated with the following **attribute-types**:

1. **ATTRIBUTE-TYPE NAME = SCHEMA-PERMISSION-1**

PURPOSE - "The **SCHEMA-PERMISSION-1** attribute-type, when assigned the value **YES**, will assign to this dictionary user permission to execute **all** the commands available for interaction with the dictionary schema specified in Chapter 6, as well as the **EXTRACT-ESSENTIAL-SCHEMA** and **COMPARE-ESSENTIAL-SCHEMAS** commands of Chapter 7. The default value is **NO**."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

2. **ATTRIBUTE-TYPE NAME = SCHEMA-PERMISSION-2**

PURPOSE - "The **SCHEMA-PERMISSION-2** attribute-type, when assigned the value **YES**, will assign to this dictionary user permission to execute **all** the commands, except the **ABOLISH-META-ENTITY-WITH-LOCK** and **ALTER-META-ENTITY-WITH-LOCK** commands, which are available for interaction with the dictionary schema specified in Chapter 6, as well as the **EXTRACT-ESSENTIAL-SCHEMA** and **COMPARE-ESSENTIAL-SCHEMAS** commands of Chapter 7. The default value is **NO**."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

3. ATTRIBUTE-TYPE NAME = SCHEMA-PERMISSION-3

PURPOSE - "The SCHEMA-PERMISSION-3 attribute-type, when assigned the value YES, will assign to this dictionary user permission to execute **all reporting commands** for interaction with the dictionary schema specified in Section 5.2, as well as the EXTRACT-ESSENTIAL-SCHEMA and COMPARE-ESSENTIAL-SCHEMAS commands of Chapter 7. The default value is NO."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

4. ATTRIBUTE-TYPE NAME = SCHEMA-PERMISSION-4

PURPOSE = "The SCHEMA-PERMISSION-4 attribute-type, when assigned the value YES, will assign to this dictionary user permission to execute the commands

ALTER-META-ENTITY
META-LIST

on meta-entities of type ATTRIBUTE-TYPE-
VALIDATION-DATA."

BASIC/EXTENDED = BASIC

SYSTEM/LOCK = ON

5. ATTRIBUTE-TYPE NAME = SCHEMA-PERMISSION-5

PURPOSE = "The SCHEMA-PERMISSION-5 attribute-type, when assigned the value YES, will assign to this dictionary user permission to execute

the command META-LIST on meta-entities of
type ATTRIBUTE-TYPE-VALIDATION-DATA."

BASIC/EXTENDED = BASIC

SYSTEM/LOCK = ON

6. ATTRIBUTE-TYPE NAME = WRITE-KEY

PURPOSE = "The WRITE-KEY attribute-type serves to
assign permission to this dictionary user to
execute commands on an entity in the
dictionary which has local protection with a
matching WRITE-LOCK attribute."

BASIC/EXTENDED = BASIC

SYSTEM/LOCK = ON

7. ATTRIBUTE-TYPE NAME = READ-KEY

PURPOSE = "The READ-KEY attribute-type serves to
assign permission to this dictionary user to
execute commands that read an entity in the
dictionary which has local protection with a
matching READ-LOCK attribute."

BASIC/EXTENDED = BASIC

SYSTEM/LOCK = ON

8. ATTRIBUTE-TYPE NAME = EXCLUDE-RELATIONSHIPS

PURPOSE = "The EXCLUDE-RELATIONSHIPS attribute-type
serves to state the names of relationship-
types in the schema, the presence of which
are hidden from this user. An attribute of
this type must be the name of a relation-
ship-type."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

9. ATTRIBUTE-TYPE NAME = DEFAULT-STATUS-NAME

PURPOSE = "The DEFAULT-STATUS-NAME attribute-type serves to specify the name of the status that exists as the default for a user of the dictionary system."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = ON

8.1.2 RULES FOR THE ENTITY-TYPE DICTIONARY-USER

The following rules apply to the DICTIONARY-PERMISSIONS attribute-group-type and the other attribute-types associated with the DICTIONARY-USER entity-type:

1. In the DICTIONARY-PERMISSIONS attribute-type, the meaning of the attribute-types is as follows:
 - (a) The value of the attribute-type STATUS declares the status for which permissions are being assigned. The value *ALL designates that permissions are being established for all statuses.
 - (b) The name of an entity-type specified declares that permissions are being established for entities of that type in the status named.
2. For any command which involves more than one entity, such as ADD-RELATIONSHIP, execution of the command requires appropriate permissions for all entities involved.
3. The CHANGE-STATUS command requires STATUS-PERMISSION

in both of the statuses named in the command.

4. ADD-PERMISSION, DELETE-PERMISSION, or MODIFY-PERMISSION cannot be established for the status whose name is CONTROLLED.
5. The status declared for PROTECT-PERMISSION is the status in which the entities reside for which local security is being established, deleted, or modified.
6. Invocation of the security clause in the ADD-ENTITY command requires PROTECT-PERMISSION for the entity-type in the status which is addressed in the command.
7. Adding, deleting, or modifying entities of type DICTIONARY-USER requires ADMINISTRATOR-PERMISSION in the status named SECURITY-STATUS.
8. Inclusion of the entity-type ACCESS-CONTROLLER in an attribute-group of type DICTIONARY-PERMISSIONS has no meaning and will be ignored. No commands exist that require direct access permission to such entities, as the entity-type declared for PROTECT-PERMISSION is the entity-type of the entities that are being controlled.
9. Multiple attribute-groups of type DICTIONARY-PERMISSIONS are allowed.
10. If the name of a relationship-type is specified in an attribute of type EXCLUDE-RELATIONSHIPS, this relationship-type cannot be referenced by the user in any command.
11. Entities of type DICTIONARY-USER cannot be members of any relationship.

8.1.3 ACCESS-CONTROLLER ENTITY-TYPE

Entities of type **ACCESS-CONTROLLER** serve to provide the specification of local permissions for users of a dictionary system. The definition of this entity-type given in Chapter 4. is:

ENTITY-TYPE NAME = **ACCESS-CONTROLLER**

PURPOSE = "To specify access restrictions to an entity or set of entities in the dictionary. Entities of this type are used exclusively in the security facility of the dictionary system."

BASIC/EXTENDED = BASIC

SYSTEM-LOCK = OFF

DATE-CREATED-IN-SCHEMA

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

CREATED-IN-SCHEMA-BY

will have an implementor-defined value showing that this descriptor is part of the system-standard schema.

DATE-LAST-MODIFIED-IN-SCHEMA

will have a null value.



LAST-MODIFIED-IN-SCHEMA-BY
will have a null value.

NUMBER-OF-TIMES-MODIFIED-IN-SCHEMA = "0"

ENTITY-TYPE = SECURITY

The following attribute-types are associated with this entity-type:

1. ATTRIBUTE-TYPE NAME = **WRITE-LOCK**

PURPOSE = "The WRITE-LOCK attribute-type serves to provide protection against unauthorized commands that attempt to write on a protected entity. Commands that write on such a protected entity require a matching WRITE-KEY in the DICTIONARY-USER entity."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

2. ATTRIBUTE-TYPE NAME = **READ-LOCK**

PURPOSE = "The READ-LOCK attribute-type serves to provide protection against unauthorized commands that attempt to read a protected entity. Commands that read such a protected entity require either a matching READ-KEY or WRITE-KEY in the DICTIONARY-USER entity."

BASIC/EXTENDED = BASIC

SYSTEM-GENERATED = YES

SYSTEM-LOCK = ON

8.1.4 RULES FOR THE ENTITY-TYPE ACCESS-CONTROLLER

The following rules apply to the entity-type ACCESS-CONTROLLER:

1. The entity-type ACCESS-CONTROLLER is a member of relationship-types, each entity-type in the schema (with exception of the entity-type DICTIONARY-USER) being the other member of one of these relationship-types. Whenever the CREATE-META-ENTITY command is invoked and the meta-entity-type ENTITY-TYPE is specified, execution of this command will also create a relationship-type whose members are the entity-type being created and the entity-type ACCESS-CONTROLLER.
2. Whenever a relationship between an entity of type ACCESS-CONTROLLER and a version of an entity in the dictionary is created, should this entity have other versions, relationships with all other versions are also created. This means that all versions of an entity have the same local security protection.

8.1.5 EFFECTS OF THE DDS SECURITY FACILITY

The effects of the security facility will be discussed in terms of the commands that are to be executed.

8.1.5.1 QUALIFICATION COMMANDS

The execution of a qualification command will proceed as if the dictionary user had permission to access the entire dictionary. When the count of the entities selected is returned to the user, this count is returned in terms of the number of such entities that are accessible to the user, and the number of entities for which the user does not have permission, either because of global or local security restrictions.

In case a dictionary user requests a listing of a qualification

list, in this listing the primary name and all attributes of every entity not accessible to that user will be replaced by a series of asterisks or some other implementor defined set of literals.

8.1.5.2 MAINTENANCE COMMANDS

The effect of the security facility on maintenance commands is as follows:

1. The ADD-ENTITY command requires ADD-PERMISSION in the status where the entity is to be added and for the entity-type of the new entity. Additionally, if the new entity is to be protected with a security clause in the command, PROTECT-PERMISSION is required.
2. The ADD-RELATIONSHIP command requires ADD-PERMISSION for both of the entities which are members of the relationship. If an implicit entity is being created, ADD-PERMISSION is required for all the potential entity-types that apply to the implicit entity. If either of the entities has local protection, the WRITE-KEY for that entity is required.
3. The CHANGE-STATUS command requires STATUS-PERMISSION for both statuses specified in the command. If an entity specified has local protection, the WRITE-KEY is required.
4. The COPY command requires ADD-PERMISSION for the status and entity-type specified. If the entity which is the source has local protection, the READ-KEY is required.
5. The DECLARE command requires ADD-PERMISSION for the status and entity-type specified.
6. The DELETE-ENTITY command requires DELETE-PERMISSION for the status and entity-type specified. If the entity has local protection, the WRITE-KEY is required.

7. The DELETE-RELATIONSHIP command requires DELETE-PERMISSION for the status and entity-type of both entities which are members of the relationship. If an entity has local protection, the WRITE-KEY for that entity is required.
8. The MODIFY-ENTITY command requires MODIFY-PERMISSION for the status and entity-type of the entity. If the entity has local protection, the WRITE-KEY for the entity is required.
9. The MODIFY-RELATIONSHIP command requires MODIFY-PERMISSION for the status and entity-type of both entities which are members of the relationship. If an entity has local protection, the WRITE-KEY for that entity is required.
10. The RENAME command requires ADD-PERMISSION for the status and entity-type of the entity specified. If the entity has local protection, the WRITE-KEY is required.
11. The RENUMBER command requires MODIFY-PERMISSION for the status and entity-type of an entity specified. If local protection exists, the WRITE-KEY is required.

8.1.5.3 REPORT COMMANDS

In a manner analogous to qualification commands, if a dictionary user requests a report in which entities would appear which are not accessible to that user, in the report the primary name and all attributes of every entity not accessible to that user will be replaced by a series of asterisks or some other implementor defined set of literals. Similarly, attributes of relationships will be replaced by such literals whenever both members of the relationship are not accessible to the user.

8.1.5.4 QUERY COMMANDS

The effect of the security facility on query commands is identical to what has been discussed for qualification and report commands. Separate counts for entities which are accessible and which are not accessible are returned to the user. In the output of a query, the primary name and all attributes of every entity not accessible to that user will be replaced by a series of asterisks or some other implementor defined set of literals. Similarly, attributes of relationships will be replaced by such literals whenever both members of the relationship are not accessible to the user.

8.1.5.5 DDS SOFTWARE INTERFACE COMMANDS

Execution of all the DDS software interface commands that interact with a dictionary, i.e.

```
GENERATE-STRUCTURE-FOR-COBOL
EXTRACT-SUBSET
LOAD-DICTIONARY
IMPORT-SUBSET
CALL DDS
```

require access to all dictionary descriptors involved in the command.

The CREATE-DICTIONARY command requires ADMINISTRATOR-PERMISSION.

The EXTRACT-ESSENTIAL-SCHEMA and COMPARE-ESSENTIAL-SCHEMAS commands require either SCHEMA-PERMISSION-1 or SCHEMA-PERMISSION-2.

8.2 LOCAL SECURITY COMMANDS

In addition to the facility already specified as part of the ADD-ENTITY command, whereby an entity may be assigned local security at the time it is added to the dictionary, the following commands are used to establish, modify, or delete local security:

ADD-SECURITY
DELETE-SECURITY
MODIFY-SECURITY
ASSIGN-KEY
DELETE-KEY

Execution of these commands is subject to the dictionary user having been assigned PROTECT-PERMISSION for the status and entity-type of every entity being specified in a command.

8.2.1 ADD-SECURITY COMMAND

PURPOSE: To assign local security to one or more existing unprotected entities.

FORMAT: ADD-SECURITY
 {EXISTING-CONTROLLER|NEW-CONTROLLER}controller-name
 list-name

where list-name is one of the following:

one or more names which are either the primary name
or SHORT-NAME of an entity

the name of a qualification list

a RUN command with the name of a procedure

RULES:

1. If EXISTING-CONTROLLER is specified, controller-name must be the primary name of an entity of type ACCESS-CONTROLLER.
2. If NEW-CONTROLLER is specified, controller-name cannot be the primary name of an entity in the dictionary.
3. No entity in list-name can be the member of a relationship-type whose other member is an entity of type ACCESS-CONTROLLER.

ACTIONS PERFORMED:

1. If NEW-CONTROLLER is specified, an entity of type ACCESS-CONTROLLER whose name is controller-name, is added to the dictionary.
2. For every entity in list-name a relationship is created, the other member of which is the specified entity of type ACCESS-CONTROLLER.

ERRORS:

1. EXISTING-CONTROLLER is specified, and controller-name is not the primary name of an entity of type ACCESS-CONTROLLER.
2. NEW-CONTROLLER is specified, and controller-name is the primary name of an entity in the dictionary.
3. A name is specified in list-name which is not the primary name or SHORT-NAME of an entity in the dictionary.
4. A non-existent qualification list is specified.
5. A non-existent procedure is specified.
6. An entity is specified in list-name for which local protection already exists.

EXAMPLES:

1. ADD-SECURITY
EXISTING-CONTROLLER PAYROLL-CONTROLLER
NEW-SALARY, \$NEW-DT
2. ADD-SECURITY
NEW-CONTROLLER FIN148
FINANCIAL-LIST

8.2.2 DELETE-SECURITY COMMAND

PURPOSE: To delete existing local protection for one or more entities.

FORMAT: DELETE-SECURITY
list-name

where list-name is one of the following:

one or more names which are either the primary name
or SHORT-NAME of an entity

the name of a qualification list

a RUN command with the name of a procedure

RULES:

1. Every entity specified in list-name must have local protection.

ACTIONS PERFORMED:

1. The local protection of every entity in list-name is deleted.

2. If, at the completion of the deletion of local protection, there exists an entity of type ACCESS-CONTROLLER which is not the member of any relationship, this entity is also deleted.

ERRORS:

1. A name is specified in list-name which is not the primary name or SHORT-NAME of an entity in the dictionary.
2. A non-existent qualification list is specified.
3. A non-existent procedure is specified.
4. An entity is specified in list-name for which no local protection exists.

EXAMPLES:

1. DELETE-SECURITY
ELEMENT-A190
2. DELETE-SECURITY
RUN OBSOLETE-DATA-PROCEDURE

8.2.3 MODIFY-SECURITY COMMAND

PURPOSE: To remove local protection for one or more entities from existing controller(s) and to assign them to another controller. READ-KEYS and WRITE-KEYS are optionally updated.

FORMAT: MODIFY-SECURITY
list-name
TO {EXISTING-CONTROLLER|NEW-CONTROLLER}controller-name
[NEW-KEYS]

where list-name is one of the following:

one or more names which are either the primary name
or SHORT-NAME of an entity

the name of a qualification list

a RUN command with the name of a procedure

RULES:

1. Every entity in list-name must have local protection.
2. If EXISTING-CONTROLLER is specified, controller-name must be the primary name of an entity of type ACCESS-CONTROLLER.
3. If NEW-CONTROLLER is specified, controller-name cannot be the primary name of an entity in the dictionary.

ACTIONS PERFORMED:

1. The existing local protection of every entity in list-name is deleted.
2. Local protection for every entity in list-name is established with the entity whose primary name is controller-name.
3. If, at the completion of the deletion of local protection, there exists an entity of type ACCESS-CONTROLLER which is not the member of any relationship, this entity is also deleted.
4. If NEW-KEYS is specified, READ-KEY and WRITE-KEY attributes for the new controller are assigned to all DICTIONARY-USER entities which had access to the entities in list-name prior to the execution of the command.

ERRORS:

1. EXISTING-CONTROLLER is specified, and controller-name is not the primary name of an entity of type ACCESS-CONTROLLER.
2. NEW-CONTROLLER is specified, and controller-name is the primary name of an entity in the dictionary.
3. A name is specified in list-name which is not the primary name or SHORT-NAME of an entity in the dictionary.
4. A non-existent qualification list is specified.
5. A non-existent procedure is specified.
6. An entity is specified in list-name for which local protection does not exist.

EXAMPLES:

1. MODIFY-SECURITY
FINANCIAL-LIST
TO EXISTING-CONTROLLER
FINANCIAL-CONTROLLER

This command causes local protection of all entities in FINANCIAL-LIST to be transferred to the entity of type ACCESS-CONTROLLER named FINANCIAL-CONTROLLER.

NEW-KEYS has not been specified, and hence access permissions to these entities is restricted to those dictionary users for which keys to FINANCIAL-CONTROLLER have been previously assigned.

2. MODIFY-SECURITY
ELEMENT-A, ELEMENT-B
TO NEW-CONTROLLER
CONTROLLER-AC7656
NEW-KEYS

This command created a new entity of type ACCESS-CONTROLLER named CONTROLLER-AC7656 and assigns local protection of ELEMENT-A and ELEMENT-B to it. Since NEW-KEYS has been specified, all dictionary users who previously had access to those entities continue to do so.

8.2.4 ASSIGN-KEY COMMAND

PURPOSE: To assign, or optionally reassign, read and write keys to sets of dictionary users.

FORMAT:

```
ASSIGN-KEYS
access-controller-list
WRITE-KEYS TO dictionary-user-list-1 [ONLY]
READ-KEYS TO dictionary-user-list-2 [ONLY]
```

where access-controller-list is one of the following:

one or more names which are the primary name
of an entity of type ACCESS-CONTROLLER

the name of a qualification list

a RUN command with the name of a procedure

and where dictionary-user-list-1 and dictionary-user-list-2 are one of the following:

one or more names which are the primary name
of an entity of type DICTIONARY-USER

the name of a qualification list

a RUN command with the name of a procedure

RULES:

1. Every name in access-controller-list must be the primary name of an entity of type ACCESS-CONTROLLER.
2. Every name in dictionary-user-list-1 and dictionary-user-list-2 must be the primary name of an entity of type DICTIONARY-USER.

ACTIONS PERFORMED:

1. If the ONLY clause is not specified for WRITE-KEYS, then for every entity which has local protection through an ACCESS-CONTROLLER in access-controller-list, WRITE-KEY attributes are assigned to all entities of type DICTIONARY-USER in dictionary-user-list-1. Other entities of this type which already have such WRITE-KEY attributes continue to have access to these entities.
2. If the ONLY clause is not specified for READ-KEYS, then for every entity which has local protection through an ACCESS-CONTROLLER in access-controller-list, READ-KEY attributes are assigned to all entities of type DICTIONARY-USER in dictionary-user-list-1. Other entities of this type which already have such READ-KEY attributes continue to have access to these entities.
3. If the ONLY clause is specified for WRITE-KEYS, then for every entity which has local protection through an ACCESS-CONTROLLER in access-controller-list, WRITE-KEY attributes are assigned to all entities of type DICTIONARY-USER in dictionary-user-list-1. Other entities of this type which prior to the command had such WRITE-KEY attributes, no longer have access to these entities.
4. If the ONLY clause is specified for READ-KEYS, then for every entity which has local protection through an ACCESS-CONTROLLER in access-controller-list, READ-KEY attributes are assigned to all entities of type DIC-

TIONARY-USER in dictionary-user-list-1. Other entities of this type which prior to the command had such READ-KEY attributes, no longer have access to these entities.

ERRORS:

1. A name is specified in access-controller-list which is not the primary name of an entity of type ACCESS-CONTROLLER in the dictionary.
2. A non-existent qualification list for access-controller-list is specified.
3. A non-existent procedure for access-controller-list is specified.
4. A name is specified in dictionary-user-list-1 or dictionary-user-list-2 which is not the primary name of an entity of type DICTIONARY-USER in the dictionary.
5. A non-existent qualification list for dictionary-user-list-1 or dictionary-user-list-2 is specified.
6. A non-existent procedure for dictionary-user-list-1 or dictionary-user-list-2 is specified.

EXAMPLES:

1. ASSIGN-KEYS
CLASSIFIED-CONTROLLER
WRITE-KEYS TO USER-LIST-1 ONLY
READ-KEYS TO USER-LIST-2 ONLY

This command assigns WRITE-KEYS and READ-KEYS to dictionary users in USER-LIST-1 and USER-LIST-2, respectively, for all entities controlled by CLASSIFIED-CONTROLLER. This assignment is exclusive in the sense that any access permissions other users may have had prior to the command have been deleted.

2. ASSIGN-KEYS
FINANCIAL-CONTROLLER
WRITE K.JONES

This command assigns a WRITE-KEY to the entities controlled by FINANCIAL-CONTROLLER to K.JONES. Other users who have access to these entities are not affected.

8.2.5 DELETE-KEY COMMAND

PURPOSE: To delete existing READ-KEYS and WRITE-KEYS from a list of dictionary users.

FORMAT: DELETE-KEY
access-controller-list
dictionary-user-list

where access-controller-list is one of the following:

one or more names which are the primary name
of an entity of type ACCESS-CONTROLLER

the name of a qualification list

a RUN command with the name of a procedure

and where dictionary-user-list is one of the following:

one or more names which are the primary name
of an entity of type DICTIONARY-USER

the name of a qualification list

a RUN command with the name of a procedure

RULES:

1. Every name in access-controller-list must be the primary name of an entity of type ACCESS-CONTROLLER.
2. Every name in dictionary-user-list must be the primary name of an entity of type DICTIONARY-USER.

ACTIONS PERFORMED:

1. All READ-KEYS and WRITE-KEYS of the dictionary users in dictionary-user-list to the access-controllers in access-controller-list are deleted.

ERRORS:

1. A name is specified in access-controller-list which is not the primary name of an entity of type ACCESS-CONTROLLER in the dictionary.
2. A non-existent qualification list for access-controller-list is specified.
3. A non-existent procedure for access-controller-list is specified.
4. A name is specified in dictionary-user-list which is not the primary name of an entity of type DICTIONARY-USER in the dictionary.
5. A non-existent qualification list for dictionary-user-list is specified.
6. A non-existent procedure for dictionary-user-list is specified.

EXAMPLE:

DELETE-KEY
ALL-CONTROLLERS-LIST
TERMINATED-EMPLOYEES-LIST

8.3 DICTIONARY ADMINISTRATOR TOOLS

This section deals with those special tools needed by the Dictionary Administrator staff in their duties in setting up and controlling the dictionary. Because many of these tools require system dependent actions to complete their functions, and because the way in which these actions are initiated therefore is likely to have system dependencies, the material here is **not** given as a set of possible commands. Thus this section may be considered as a set of statements of requirements for tools to aid in the dictionary administration process.

The overall need for dictionary administration are:

1. To set up the dictionary databases so that different classes of users may potentially see different dictionaries -- this is particularly important in large organization applications and distributed environments.
2. To ensure continuity of service over time. This generally means that some copy of the dictionary database must be taken and that a means is available for starting the dictionary with this copy available.
3. To provide means for monitoring the performance of the dictionary system, and to have the ability to correct any possible reasons for deterioration of performance (such as poor storage utilization).

8.3.1 TOOLS FOR DICTIONARY SET UP

There are two basic assumptions in this discussion:

1. The command structure for such operations as log-in or sign-on will conform to any FIPS interface standard.

2. The dictionary administrator will be able to create more than one database, thereby producing multiple "dictionaries" for user access. Thus the sign on of the user must identify which dictionary is required by naming it.

The tools for dictionary set-up must therefore have the following characteristics:

1. The dictionary administrator must be able to create a new dictionary. This involves the ability to assign a unique name to it. The CREATE-DICTIONARY command of Section 7.2.4.2 addresses this requirement.
2. The dictionary administrator must have the option of either taking the system standard dictionary schema, or copying a given (previously named and defined) dictionary schema to be the new dictionary schema.
3. The dictionary database must either have space preallocated to it and be able to have space added to it (when running out), or else it must be able to get space when needed. This mechanism is obviously operating system and command language dependent; however, the dictionary administrator must be able to make any required actions to obtain space for each of the databases.

8.3.2 TOOLS FOR DICTIONARY CONTINUITY

Because the dictionary database may be damaged due to physical deterioration, user or software error, or other cause, there must be some way to reconstruct the dictionary as of some previous time. There is also the possibility of communications that causes multiple copies of a dictionary (e.g. at different geographical locations) to be out of synchronization -- possibly due to loss of update messages. The tools for continuity must therefore provide services that allow the dictionary database at a location to be reconstructed to its condition at some previous

time. This may be accomplished by any vendor implemented technique, but probably entails some technique such as:

1. Being able to initiate a request to make a copy of the current dictionary database and loading this copy back into the dictionary database after some disaster or major error; or
2. Being able to request that a copy of a dictionary database at one location is transferred to and loaded at another location.

It may be assumed that some vendors may make use of their operating system or database management system facilities to "restart" the system after a hardware failure, to "recover" from a failure automatically, or to "back-out" a bad transaction. However, there is no requirement for such tools in this specification.

8.3.3 TOOLS FOR PERFORMANCE IMPROVEMENT

The future system may be self organizing, in the sense that it constantly monitors performance and makes any necessary improvements that will improve the performance or detection of deterioration, however, today's systems do not generally provide such automatic facilities. Thus the system will generally be provided with two types of tools:

1. Tools to monitor performance.
2. Tools to improve performance.

Typical tools to be provided by the vendor might be expected to include:

1. Methods to collect statistics that will aid in performance assessment. These will generally be: simple collection mechanisms that determine the time of access for various dictionary maintenance and reporting commands; mechanisms for assessing the excess space utilization (e.g., in any overflowed

records); and statistics packages to work on this data to show average values and/or trends, etc.

2. Tools to aid in improving performance. These will generally not need to be highly sophisticated; e.g., they may be means to make a logical dump and restore of the database, or they may merely allow a dump output and then read-in of the file to remove any major problems due to excess overflow of records and parts of records in the physical storage devices.



APPENDIX A

FEASIBILITY OF EXTENSIBILITY IMPLEMENTATION

In the discussion on the issue as to whether to include full extensibility facilities in the proposed Federal Information Processing Standard for a DDS, or only some subset, the critical point centered on the required size of the host computer that would be running the DDS. There was a general agreement that the current state-of-the-art was to provide full facilities of this kind, but that inclusion of these facilities should not restrict the standard to be operable only on medium- to large-size computer systems.

This appendix addresses the feasibility of implementing a full extensibility facility on a small computer system. It must be emphasized that the implementation scheme which will be discussed is merely an example of an implementation, and that no claim, implicit or explicit, is being made that this scheme is a good scheme, or that it should be used in an actual implementation of the standard.

There exist in today's market-place a number of small database management systems that are available on mini and micro systems. Such database management systems support hierarchic, or network, or relational databases. Examples are:

- o HDBS (Hierarchical) by Micro Data Base Systems, Inc., which runs on Z-80, Z-8000, or Z-8080/86 based hardware under the CP/M operating system;
- o SEED (Network) by International Data Base Systems, Inc., which runs on DEC VAX-11, PDP-11, and Z-80 processors;
- o ORACLE (Relational) by RSI for DEC VAX-11 and PDP-11 systems;

- o TOTAL (Two level network) by Cincom Systems, Inc., which is available on a variety of small systems.
- o An impact can also be expected from promising new integrated systems, such as Britton Lee's IDM 500 (Relational).

Because of the relative simplicity of the Relational Database Management System (R-DBMS), it was chosen as a means of demonstrating the desired feasibility. Many such systems already contain a primitive dictionary system to store the R-DBMS schema; e.g., in IBM's SQL/DS there are many system tables that can be accessed by privileged users to provide the management and security controls of the system and to define and modify the schema.

The proof of feasibility being offered here consists of the following:

We will assume that there exist certain generically defined tables that make up the dictionary schema, and that there exists a dictionary processing system that operates on these tables.

We will then show the actions that must be performed on these tables when the extensibility commands that

1. create a new entity-type,
2. create a new relationship-type,
3. create new attribute-types, that are associated with an old or new entity-type, and
4. create a new attribute-type that is associated with a relationship-type,

are executed.

Upon completion of these executions, it will then be seen that the tables are generically identical to the ones prior to the execution of the commands.

Dictionary Tables

The following three tables exist:

1. The M-ENTITY table
2. The M-ATTRIBUTE table
3. The M-RELATIONSHIP table

Rows in the M-ENTITY table correspond to meta-entities that exist in the schema. The table has two columns - the first one is the meta-entity-type of an row, and the second one is the name of the meta-entity.

In the following we will use as an example a dictionary schema which is composed of the following:

The entity-types FILE and RECORD;

The relationship-type CONTAINS, whose first member is FILE and whose second member is RECORD;

The attribute-types NUMBER-OF-RECORDS and ACCESS-METHOD, which are associated with the entity-type FILE, and the attribute-type RESPONSIBILITY, which is associated with the entity-type RECORD.

The attribute-type DESCRIPTOR which is associated with the relationship-type CONTAINS.

Figure A-1 shows the M-ENTITY table for this schema. All figures are found at the end of this appendix.

The M-ATTRIBUTE table shows the association of attribute-types with entity-types and relationship-types. In this table there exists one row for each attribute-type in the schema. The table has two columns, the first column shows the name of the meta-entity with which the attribute-type in the second column is associated. Figure A-2 shows the M-ATTRIBUTE table for the example schema.

The M-RELATIONSHIP table shows the entity-types that participate in each relationship-type. There exists one row for each relationship-type in the schema. The table has three columns; the first column contains the name of the relationship-type, the second column contains the name of the entity-type which is the first member of the relationship, and the third column contains the name of the entity-type which is the second member. Figure A-3 shows the M-RELATIONSHIP table for the example schema.

Effect of Extensibility on Dictionary Tables

Suppose it is desired to extend the example schema by the following actions:

1. The entity-type REPORT is to be created.
2. The relationship-type USES is to be created, the first member of this relationship-type being REPORT, and the second member being RECORD.
3. The attribute-type CENTRAL-ID is to be created, and this attribute-type is to be associated with the entity-type REPORT.
4. The attribute-type PROCESSING-OPTION is to be created, and this attribute-type is to be associated with the relationship-type USES.
5. The attribute-type REVIEW-DATE is to be created, and this attribute-type is to be associated with the entity-type RECORD, which existed in the original example schema.

We will discuss the effect of each one of these actions on the dictionary schema tables. Modifications of the tables will be highlighted with bold lettering.

Execution of the **first** one of this actions adds a row in the M-ENTITY table, as is shown in Figure A-4. No modification to the other tables takes place.

Execution of the **second** action adds a row in the M-ENTITY table, as shown in Figure A-5, as well as a row in the M-RELATIONSHIP table, as shown in Figure A-6.

Execution of the **third** action adds a row in the M-ENTITY table, as shown in Figure A-7, as well as a row in the M-ATTRIBUTE table, as shown in Figure A-8.

Execution of the **fourth** action adds a row in the M-ENTITY table, as shown in Figure A-9, as well as a row in the M-ATTRIBUTE table, as shown in Figure A-10.

Finally, execution of the **fifth** and last action again adds a row in the M-ENTITY table, as shown in Figure A-11, and a row in the M-ATTRIBUTE table, as shown in Figure A-12.

Conclusion

It can then be seen that the dictionary schema tables that have been arrived at (i.e. the M-ENTITY table in Figure A-11, the M-ATTRIBUTE table in Figure A-12, and the M-RELATIONSHIP table in Figure A-6), are generically no different from the tables that described the schema prior to the modifications. Thus any operation that could be performed by the dictionary processing system on the old schema, or as a result of the old schema (e.g., populating the dictionary) is still valid on the new schema. We can therefore conclude that the system can be made fully extensible.

META-ENTITY-TYPE	META-ENTITY-NAME
ENTITY-TYPE	FILE
ENTITY-TYPE	RECORD
RELATIONSHIP-TYPE	CONTAINS
ATTRIBUTE-TYPE	NUMBER-OF-RECORDS
ATTRIBUTE-TYPE	ACCESS-METHOD
ATTRIBUTE-TYPE	RESPONSIBILITY
ATTRIBUTE-TYPE	DESCRIPTOR

Figure A-1 - M-ENTITY table for example schema

META-ENTITY-NAME	ATTRIBUTE-TYPE-NAME
FILE	NUMBER-OF-RECORDS
FILE	ACCESS-METHOD
RECORD	RESPONSIBILITY
CONTAINS	DESCRIPTOR

Figure A-2 - M-ATTRIBUTE table for example schema

RELATIONSHIP-TYPE-NAME	FROM	TO
CONTAINS	FILE	RECORD

Figure A-3 - M-RELATIONSHIP table for example schema

META-ENTITY-TYPE	META-ENTITY-NAME
ENTITY-TYPE	FILE
ENTITY-TYPE	RECORD
RELATIONSHIP-TYPE	CONTAINS
ATTRIBUTE-TYPE	NUMBER-OF-RECORDS
ATTRIBUTE-TYPE	ACCESS-METHOD
ATTRIBUTE-TYPE	RESPONSIBILITY
ATTRIBUTE-TYPE	DESCRIPTOR
ENTITY-TYPE	REPORT

Figure A-4 - M-ENTITY table after Action 1

META-ENTITY-TYPE	META-ENTITY-NAME
ENTITY-TYPE	FILE
ENTITY-TYPE	RECORD
RELATIONSHIP-TYPE	CONTAINS
ATTRIBUTE-TYPE	NUMBER-OF-RECORDS
ATTRIBUTE-TYPE	ACCESS-METHOD
ATTRIBUTE-TYPE	RESPONSIBILITY
ATTRIBUTE-TYPE	DESCRIPTOR
ENTITY-TYPE	REPORT
RELATIONSHIP-TYPE	USES

Figure A-5 - M-ENTITY table after Action 2

	RELATIONSHIP-TYPE-NAME		FROM		TO	
	CONTAINS		FILE		RECORD	
	USES		REPORT		RECORD	

Figure A-6 - M-RELATIONSHIP table after Action 2

	META-ENTITY-TYPE		META-ENTITY-NAME	
	ENTITY-TYPE		FILE	
	ENTITY-TYPE		RECORD	
	RELATIONSHIP-TYPE		CONTAINS	
	ATTRIBUTE-TYPE		NUMBER-OF-RECORDS	
	ATTRIBUTE-TYPE		ACCESS-METHOD	
	ATTRIBUTE-TYPE		RESPONSIBILITY	
	ATTRIBUTE-TYPE		DESCRIPTOR	
	ENTITY-TYPE		REPORT	
	RELATIONSHIP-TYPE		USES	
	ATTRIBUTE-TYPE		CENTRAL-ID	

Figure A-7 - M-ENTITY table after Action 3

META-ENTITY-NAME	ATTRIBUTE-TYPE-NAME
FILE	NUMBER-OF-RECORDS
FILE	ACCESS-METHOD
RECORD	RESPONSIBILITY
CONTAINS	DESCRIPTOR
REPORT	CENTRAL-ID

Figure A-8 - M-ATTRIBUTE table after Action 3

META-ENTITY-TYPE	META-ENTITY-NAME
ENTITY-TYPE	FILE
ENTITY-TYPE	RECORD
RELATIONSHIP-TYPE	CONTAINS
ATTRIBUTE-TYPE	NUMBER-OF-RECORDS
ATTRIBUTE-TYPE	ACCESS-METHOD
ATTRIBUTE-TYPE	RESPONSIBILITY
ATTRIBUTE-TYPE	DESCRIPTOR
ENTITY-TYPE	REPORT
RELATIONSHIP-TYPE	USES
ATTRIBUTE-TYPE	CENTRAL-ID
ATTRIBUTE-TYPE	PROCESSING-OPTION

Figure A-9 - M-ENTITY table after Action 4

META-ENTITY-NAME	ATTRIBUTE-TYPE-NAME
FILE	NUMBER-OF-RECORDS
FILE	ACCESS-METHOD
RECORD	RESPONSIBILITY
CONTAINS	DESCRIPTOR
REPORT	CENTRAL-ID
USES	PROCESSING-OPTION

Figure A-10 - M-ATTRIBUTE table after Action 4

META-ENTITY-TYPE	META-ENTITY-NAME
ENTITY-TYPE	FILE
ENTITY-TYPE	RECORD
RELATIONSHIP-TYPE	CONTAINS
ATTRIBUTE-TYPE	NUMBER-OF-RECORDS
ATTRIBUTE-TYPE	ACCESS-METHOD
ATTRIBUTE-TYPE	RESPONSIBILITY
ATTRIBUTE-TYPE	DESCRIPTOR
ENTITY-TYPE	REPORT
RELATIONSHIP-TYPE	USES
ATTRIBUTE-TYPE	CENTRAL-ID
ATTRIBUTE-TYPE	PROCESSING-OPTION
ATTRIBUTE-TYPE	REVIEW-DATE

Figure A-11 - M-ENTITY table after Action 5

META-ENTITY-NAME	ATTRIBUTE-TYPE-NAME
FILE	NUMBER-OF-RECORDS
FILE	ACCESS-METHOD
RECORD	RESPONSIBILITY
CONTAINS	DESCRIPTOR
REPORT	CENTRAL-ID
RECORD	REVIEW-DATE

Figure A-12 - M-ATTRIBUTE-TABLE after Action 5

APPENDIX B

EXAMPLES OF THE USE OF META-ATTRIBUTES

In this appendix we will discuss a series of examples that deal with the Dictionary Schema and its structure. In order to better illustrate some of the points discussed, we will use some of the commands available in the core standard for interaction with the Schema, these commands being specified in detail in Chapter 5.

As is true throughout this document, the examples presented should **not** be considered to be part of the specification of the core standard.

It is again emphasized that the syntax used here is merely illustrative and that no implication whatsoever is intended that it is representative of the actual syntax to be used in the standard.

Suppose that there exists somewhere a dictionary which contains (not exclusively) the following:

FILE-A
PAYROLL-FILE
TIMESHEET-FILE

Each one of these entities has an entity-type, and as their names suggest, their entity-type has the name FILE. If we examine the dictionary further, we would find that there also exist entities with names

PAYROLL-RECORD
TIME-RECORD
A-RECORD
B-RECORD

which again, as their names suggest, are entities of the entity-type with name RECORD. We furthermore may find

entities with names

EMPLOYEE-ID
SALARY
DEPARTMENT
REPORTING-PERIOD

NO-OF-HOURS
AB1897
AC7680

.
.
.

which are all entities of the entity-type with name ELEMENT. If we examine the dictionary further we would find that the file named PAYROLL-FILE CONTAINS the record PAYROLL-RECORD, the file named TIMESHEET-FILE CONTAINS the record TIME-RECORD, and the file named FILE-A CONTAINS the records A-RECORD and B-RECORD.

Similarly, we would find that the record PAYROLL-RECORD CONTAINS (among others) the elements EMPLOYEE-ID, SALARY, and DEPARTMENT, the record TIME-RECORD CONTAINS (among others) the elements EMPLOYEE-ID, DEPARTMENT, and NO-OF-HOURS, and that A-RECORD CONTAINS the element AB1897 and that B-RECORD CONTAINS the element AC7680, among others.

We can draw some conclusions from the preceding. First of all, were we to query the schema in the following manner:

What entity-types are contained in the schema?

the response would be:

FILE
RECORD
ELEMENT

as well as some others that we have not discussed. Similarly, the query

What relationship-types are contained in the schema?

would create the following response:

FILE-CONTAINS-RECORD
RECORD-CONTAINS-ELEMENT

and, again, some others that we have not discussed. Suppose that we wished to examine one of these relationship-types closer. Since a relationship-type denotes an ordered pair of entity-types we would find that the relationship-type FILE-CONTAINS-RECORD had the entity-type FILE as the first member of the pair, and the entity-type RECORD as the second member of the pair.

We would similarly find that the dictionary would contain attributes of the entities, as well as the relationships, that are contained in it. For example, the attribute-type ACCESS-METHOD, which pertains to entities of entity-type FILE might have the value RANDOM for the files PAYROLL-FILE and TIMESHEET-FILE, and the value SEQUENTIAL for the other two files. Similarly, the attribute-type LENGTH for entities of entity-type ELEMENT would indicate the length of the instances of these entities, as they exist in the domain of information resources. This is to say that were we to examine any one of the instances of PAYROLL-RECORD, we would find that every EMPLOYEE-ID would be an 8-digit number, which is consistent with the attribute of type LENGTH for EMPLOYEE-ID being 8.

Similarly to the other queries that were directed to the schema, we could also issue a query to find out what attribute-types were in existence in the schema, and to which entity-types and/or relationship-types they pertained. Among others, the response would indicate that there was an attribute-type with name ACCESS-METHOD that pertains to entities of type FILE, and an attribute-type with name LENGTH that pertains to entities of type ELEMENT.

Suppose now that the installation to which the dictionary that we have been using as an example has developed a new requirement which they wish to support with the dictionary system. What has transpired is that it has been recognized that good documentation would substantially upgrade the entire data processing operation (as undoubtedly it would), and to this effect a new section called the "documentation center" has been created. This new section has decided that it would like to have the dictionary contain the actual location of the documentation for every file, system, program, and module. We have already seen that the

schema contains an entity-type called FILE, and we will assume that there also exist in the schema entity-types with names SYSTEM, PROGRAM, and MODULE.

Various alternatives are discussed as to how this requirement can be satisfied. On the very simplest end of the scale, a new attribute-type with name DOCUMENTATION-LOCATION could be created, and this attribute-type could be assigned to all entities of the type FILE, SYSTEM, PROGRAM, and MODULE. This alternative would seem to satisfy the basic requirement, but the documentation center staff has more extensive ideas. The proposal that they generate consists of the following:

1. Create a new entity-type called LOCATION in the dictionary, such that the names of instances of entities of this type will be the physical location of where the documentation exists.
2. Create a set of relationship-types as follows:

LOCATION-HAS-DOCUMENTATION-OF-FILE
(with members LOCATION and FILE)
LOCATION-HAS-DOCUMENTATION-OF-SYSTEM
(with members LOCATION and SYSTEM)
LOCATION-HAS-DOCUMENTATION-OF-PROGRAM
(with members LOCATION and PROGRAM)
LOCATION-HAS-DOCUMENTATION-OF-MODULE
(with members LOCATION and MODULE)

3. Create an attribute-type called MEDIUM, which is to pertain to each one of these relationship-types, which is to denote the physical medium in which the documentation exists. It is further desired that the dictionary restrict the attributes of this type to be one of the following: HARD-COPY, MICROFICHE, MICROFILM. Furthermore, the dictionary system should enforce the rule that the documentation for any one entity can only exist in one of these media.
4. Create attribute-types with names SUPERVISOR and EXTENSION, to be assigned to the entity-type LOCATION, the values of which will denote the name of the person responsible for that location and the telephone extension on which that person can be reached.

We will not try to discuss whether this proposal really is a good one or not, which in any case may depend a great deal on the installation and how they go about doing their job, as well as the people involved. Suffice it to say that we will assume that this proposal was accepted.

At this point the schema must be modified to include these new schema descriptors (schema descriptor being the generic name that we use for any component of the schema, regardless whether it is an entity-type, relationship-type, etc.). This can be achieved through the use of the CREATE-META-ENTITY and CREATE-META-RELATIONSHIP commands (Sections 5.1.8 and 5.1.9, respectively). The CREATE-META-ENTITY command is used to set up the respective descriptors and their characteristics in the schema, and the CREATE-META-RELATIONSHIP command is then used to indicate the members of relationship-types, and to assign attribute-types to entity-types and relationship-types. We will now examine these actions in more detail.

The entity-type LOCATION can be created in the schema by the command

CREATE-META-ENTITY ENTITY-TYPE LOCATION

Execution of the command will not only create this entity-type in the schema, but the dictionary system will also create along with the entity-type certain, what we call, meta-attributes. These are similar to the audit-attributes that exist in the dictionary, but are intended here to record the date on which the descriptor in question was created, who created it, and will then track all modifications made to the descriptor, such as the last date it was modified, who made the last modification, and the total number of times it was modified.

Along with the command any one of a number of optional clauses can be specified, (corresponding to the descriptors given in Section 3.1.3). Each one of these refers to what the specification calls a "meta-attribute-type", this name having been chosen since it behaves exactly like an attribute-type, but the values being specified apply to all descriptors of the type being created. Other commands exist that allow modification of these values at a later time.

Before proceeding we will discuss these "meta-attribute-types" further. In the case of attribute-types, such as ACCESS-METHOD for the entity-type FILE, we assign an attribute to every entity. For example, we have said that the attribute RANDOM applies to the FILE entity PAYROLL-FILE, and this attribute and others reside in the dictionary. On the other hand what we are talking about here are meta-attributes, and we will see in the manner in which they are used, that a single meta-attribute represents a specification that applies to all entities of a type. Among the more important of these meta-attribute-types that might be specified in the example we are dealing with are:

- o PURPOSE - This allows a string of text to be stored in the schema, that would contain the purpose that the entity-type is intended to serve. For instance, for the entity-type LOCATION it might be something like the following:

Instances of this entity-type consist of the physical locations in the documentation center at which the documentation for a file, system, program, or module is stored.

- o MINIMUM-NAME-LENGTH and MAXIMUM-NAME-LENGTH - The purpose of these meta-attribute-types is to enforce conventions that the installation wishes to adopt on the length of names that entities of a given type may have. In the case we are dealing with, let us assume that the documentation center staff has developed a coding scheme where every location name is composed of a pair of letters (denoting the coordinates of the room) and one to three numbers (denoting the shelf or drawer at that location). Then every location name must be at least three characters in length, but cannot exceed 5. This would be expressed by the clauses:

MINIMUM-NAME-LENGTH = 3
MAXIMUM-NAME-LENGTH = 5

- o PICTURE - This clause allows the alphabetic/numeric pattern or patterns of entity-names to be specified. In the example we are dealing with here, the following clause would express the requirements stated above:

PICTURE = AAN, AANN, AANNN

indicating that the name of an entity of type LOCATION must always begin with two alphabetic characters, to be followed by either one, or two, or three numbers.

- o SYSTEM-GENERATED - This meta-attribute-type, which is not of interest in the example we are dealing with, allows the specification that names of entities of this type will not be supplied by the user, but rather will be assigned by the system (according to a given PICTURE clause, and always assuring that these names are unique). This capability addresses the requirement that in certain cases it may not be meaningful to have users supply the primary names of entities of a given type, but would prefer to have these entities named with a system-generated "identifier".
- o ALTERNATE-ENTITY-TYPE-NAME - This clause addresses a requirement, which in the example we are dealing with might be that the name LOCATION that has been assigned to the entity-type might not be sufficiently descriptive for some users of the dictionary system, and might at the same time be too long for the staff of the documentation center. In this case the clause

ALTERNATE-ENTITY-TYPE-NAME =
DOCUMENTATION-LOCATION

would allow any user to refer to this entity-type by the name DOCUMENTATION-LOCATION, and the additional clause

ALTERNATE-ENTITY-TYPE-NAME = LOC

would then also provide a name requiring fewer key strokes for the staff of the documentation center.

- o ENTITY-CLASS - this meta-attribute-type is intended to record whether the entity-type being created represents DATA, PROCESS, or EXTERNAL entities. An additional value used is SECURITY, which however is

only used for entity-types that are involved with the security facility of the DDS. This meta-attribute is optional in the case we are dealing with.

This completes the specification of the entity-type LOCATION, and we can then proceed to look at the specification of one of the required relationship-types, for instance LOCATION-HAS-DOCUMENTATION-OF-SYSTEM, which has the entity-type LOCATION as its first member and the entity-type SYSTEM as its second member. The command to create the relationship-type is the same as we used before, namely

```
CREATE-META-ENTITY RELATIONSHIP-TYPE
LOCATION-HAS-DOCUMENTATION-OF-SYSTEM
```

where we indicate that we are not creating an entity-type, but a relationship-type. Again there are optional meta-attribute clauses that can be specified (corresponding to the list given in Section 3.1.3) , but looking at the ones that we used for the entity-type LOCATION we can immediately see that they really would not apply to a relationship-type the same way in which they fit an entity-type. For instance, a specification relating to length of a name is meaningful to an entity, but not to a relationship, and the same is true of the PICTURE clause.

What we have here is again an analogous situation to what we are used to seeing in the dictionary, where ACCESS-METHOD is a meaningful attribute-type for the entity-type FILE, but does not have a great deal of meaning for the entity-types ELEMENT or SYSTEM. In that same sense, then, certain meta-attribute-types are meaningful in the case where an entity-type is being created, and others are meaningful in the case of a relationship-type. As we will see shortly, other meta-attribute-types make sense when an attribute-type is being created in the schema.

This leads us to pursue this analogy further in the terminology that is being used in the specification of the schema and its structure:

In the case of the dictionary, different attribute-types are associated with each entity-type.

In the case of the schema, since different meta-attribute-types are associated with an entity-type, with a relation-

ship-type, and with an attribute-type, we will call each one of these a "meta-entity-type".

Two points need to be made for purposes of clarification. When we say that different attribute-types are associated with each entity-type, we do not wish to preclude the possibility that some attribute-types may be meaningful in the case of more than one entity-type - certainly the attribute-type DESCRIPTION is meaningful to all entity-types. Similarly, we do not wish to preclude that a certain meta-attribute-type may apply to more than one meta-entity-type.

The second point that should be made merely in passing, is that the two meta-entity-types, ENTITY-TYPE and RELATIONSHIP-TYPE, which we have dealt with in the preceding are not the only ones that exist in the structure of the system-standard schema, and later in this example we will show both the requirement for and the existence of others.

We now return to the question of what meta-attribute-types are meaningful in the creation of a relationship-type. The more important ones are the following:

- o In the same manner as for the creation of an entity-type in the schema, it is important to know when and by whom a relationship-type has been created, as well as similar data about its modifications.
- o In many instances it is desirable to assign an INVERSE NAME to a relationship-type. In the case we are dealing with, the (forward) name is LOCATION-HAS-DOCUMENTATION-OF-SYSTEM, and an instance of this relationship-type might be declared as something like

AF123 HAS-DOCUMENTATION-OF SYSTEM-R5.

In some cases it may however be desirable to make the declaration backwards, so as to be able to state something like

SYSTEM-R5 DOCUMENTATION-IS-LOCATED AT AF123

for which purpose an inverse name of the relationship-type is required.

- o There exist cases of relationship-types where more than one instance of this type can exist between the same pair of entities. A simple example is a relationship between an entity of type PROGRAMMER and an entity of type PROGRAM which is intended to denote that the programmer worked on a program during any one day. Since it is possible that the same programmer worked on a given program on more than a single day, multiple relationships can exist. Whether or not such circumstances are allowed for a given relationship-type is something that must be specified in the schema. If it is allowed, then some means must be specified for telling these various instances apart, which can be done through the proper meta-attribute-type clauses.

We will now proceed to the definition of the required attribute-types. The first one of these is the attribute-type called MEDIUM, which is to apply to the relationship-types that have been created. As was mentioned earlier, the procedure that is to be followed is to first define the attribute-type, and then to assign it to a relationship-type. The present example illustrates why the attribute-type is not being defined as part of the relationship-type definition, as in that case it would have been necessary to define it for every relationship-type to which it is to be assigned.

The CREATE-META-ENTITY command used is the same one that we have seen before, except that this time we specify the "meta-entity-type" attribute-type. Thus we have the command

CREATE-META-ENTITY ATTRIBUTE-TYPE MEDIUM

where this time the optional clauses (again from the list given in Section 3.1.3) are very similar to those used in the creation of an entity-type in that they allow specification of the following:

the minimum length of an attribute

the maximum length of an attribute

the PICTURE of an attribute

an alternate-attribute-type name

whether or not the dictionary system should generate the attribute

The reason for this last clause is that it makes it possible to define an attribute-type that can be used as an "identifier", i.e. in the case where the attribute-type is assigned to an entity-type, this attribute-type can be made to provide another unique name for each entity in addition to the primary name of the entity. This however is not the case here, and it would also appear that none of the optional clauses are of interest in the case of the attribute-type MEDIUM.

Two other attribute-types are required, namely SUPERVISOR and EXTENSION, which are to be assigned to the entity-type LOCATION. For the first one of these the command required is

CREATE-META-ENTITY ATTRIBUTE-TYPE SUPERVISOR

and in this case we do want to use some of the optional clauses.

- o The minimum length of an attribute (which consists of the name of the supervisor) is to be 5 characters.
- o The maximum length of an attribute is to be 24 characters.

For the attribute-type EXTENSION the required command is

CREATE-META-ENTITY ATTRIBUTE-TYPE EXTENSION

and we do want an optional clause that specifies that the PICTURE of an attribute must be NNNN, i.e. that every extension is a 4-digit number.

There is still more work to be done. Specifically

- o The attribute-types SUPERVISOR and EXTENSION must be assigned to the entity-type LOCATION.
- o The attribute-type MEDIUM must be assigned to each one of the relationship-types that have been created.

- o The entity-types that are members of these relationship-types must be specified, e.g. LOCATION and SYSTEM are the members of LOCATION-HAS-DOCUMENTATION-OF SYSTEM.
- o We still have not done anything to assure that the only attributes of MEDIUM can be HARD-COPY, MICROFICHE, and MICROFILM.

We will now deal with each one of these in the order listed. As was mentioned earlier, the CREATE-META-RELATIONSHIP command is used to associate an attribute-type with an entity-type. Thus we have

CREATE-META-RELATIONSHIP LOCATION AND SUPERVISOR

which is an instance of the meta-relationship-type M-R-T(ENTITY-TYPE, ATTRIBUTE-TYPE) of Section 3.1.2, and which associates the attribute-type SUPERVISOR with the entity-type LOCATION. Similar to what we have seen in the CREATE-META-ENTITY command, there can exist optional clauses as part of the command. Of particular interest are the following:

- o The ability to specify the number of attributes of the stated type that can occur for any one entity. In the case we are dealing with here, every location has a single supervisor, and this can be expressed by the clause

SINGULAR/PLURAL = SINGULAR

- o Dealing for the moment with another example, such as for an entity-type called PROGRAMMER and an attribute-type called LANGUAGE-PROFICIENCY, it is generally reasonable to assume that multiple attributes may occur, as a programmer may be familiar with several programming languages. In this case the corresponding clause would be

SINGULAR/PLURAL = PLURAL

In the case we are dealing with here it may be desirable to be able to specify some upper limit to the number of programming languages that may be specified

for any one programmer as, for instance, it may be very unlikely (and probably erroneous) to have a programmer shown who is proficient in 10 programming languages. This limitation on the number of attributes could then be expressed by the clause

MAXIMUM-NUMBER-OF-OCCURRENCES = 4

or whatever number is deemed to be a logical maximum.

In a similar manner we would assign the attribute-type EXTENSION to the entity-type LOCATION by the command

CREATE-META-RELATIONSHIP LOCATION AND EXTENSION

and assuming that every supervisor had but a single telephone extension we would specify the optional clause

SINGULAR/PLURAL = SINGULAR

We will now assign the attribute-type MEDIUM to one of the relationship-types that we have created, say, LOCATION-HAS-DOCUMENTATION-OF-SYSTEM. The required command is

CREATE-META-RELATIONSHIP
LOCATION-HAS-DOCUMENTATION-OF-SYSTEM AND MEDIUM

which is again an instance of the meta-relationship-type M-R-T(RELATIONSHIP-TYPE, ATTRIBUTE-TYPE) given in Section 3.1.2. We assume that the proper meta-attribute clause is

SINGULAR/PLURAL = SINGULAR

denoting that, if the documentation for a single system exists in more than one medium, each such piece of the documentation will be stored in a different location.

Before proceeding further, we will again digress slightly into the terminology that is being used in the specification of the core standard. This digression is not necessary from the point of view of the example we are discussing here, but does provide some help and explanation of the specification proper that is being presented. We have already introduced the concept of meta-entities, and that it was useful to consider also a "type" for

these, which we called a "meta-entity-type", and that entity-type, relationship-type, and attribute-type were meta-entity-types. In discussing the optional clauses for the CREATE-META-ENTITY command, as well as in the above cases of the CREATE-META-RELATIONSHIP command, we have seen that we are dealing with something that can reasonably be called meta-attribute-types. The question that now arises is what meaning can be assigned in this environment to, for instance, the command

CREATE-META-RELATIONSHIP LOCATION AND EXTENSION

and the meta-attribute clause that goes with it. We will look at it from the following perspective:

Both LOCATION and EXTENSION are meta-entities, and what we are doing is very similar to what we are used to doing in the dictionary -- namely, establishing a relationship between them. The only real difference is that we are operating on a different level -- in the dictionary we establish relationships between entities, and here, where we are working in the schema, we establish relationships between meta-entities. It then seems reasonable that we should call this a "meta-relationship", and indeed our terminology is very appropriate as it turns out that our meta-relationship has a meta-attribute, namely SINGULAR.

This logic can be carried one step further. In the same manner as relationships have a type, so do meta-relationships. In the case of relationships, the relationship-type involves two entity-types, and we follow the analogy by saying that a meta-relationship-type involves two meta-entity-types. In the command

CREATE-META-RELATIONSHIP LOCATION AND EXTENSION

these meta-entity-types are entity-type and attribute-type, and in the command

CREATE-META-RELATIONSHIP

LOCATION-HAS-DOCUMENTATION-OF-SYSTEM AND MEDIUM

the meta-entity-types involved are relationship-type and attribute-type.

It then also makes sense why we find that different meta-attribute clauses apply to where the CREATE-META-RELATIONSHIP command is used, as, analogous to the dictionary, different meta-relationship-types have different meta-attribute-types.

The next item which must be dealt with is the association of entity-types with the specified relationship-types. We should first note that the relationship-types that we have specified, as for instance, LOCATION-HAS-DOCUMENTATION-OF-SYSTEM, other than in the name we have given it, does not specify which entity-types are involved in the relationship-type. Every such relationship-type involves two entity-types in an ordered pair, and it will be necessary to specify which entity-type is the first member of the pair, and which one is the second one. This specification is then done through the following commands:

```
CREATE-META-RELATIONSHIP
    LOCATION-HAS-DOCUMENTATION-OF-SYSTEM AND LOCATION
```

with the additional clause

```
POSITION = FIRST
```

which denotes that the entity-type LOCATION is the first member of the relationship-type, and the command

```
CREATE-META-RELATIONSHIP
    LOCATION-HAS-DOCUMENTATION-OF-SYSTEM AND SYSTEM
```

with the clause

```
POSITION = SECOND
```

to denote that the entity-type SYSTEM is the second member of the relationship-type.

Equivalent commands are used to associate entity-types with the other relationship-types that have been specified, and the only task remaining is to include the proper provisions in the schema whereby it can be assured that the attribute-type MEDIUM can only have the attributes HARD-COPY, MICROFICHE, or MICROFILM.

There exist facilities in the core standard which we have not yet discussed in this example that allow the above specification to take place. We must first introduce the concept of what the DDS Functional Specification refers to as "attribute-type-validation-data". This is a meta-entity-type that has not yet been mentioned in this example, and its purpose is to serve as the repository of lists of legal attributes. Suppose that in the case of the attribute-type MEDIUM with which we are dealing, we decide to refer to the repository (which eventually will contain the values HARD-COPY, MICROFICHE, and MICROFILM) by the name MEDIUM-VALUES. We would then issue the command

```
CREATE-META-ENTITY
ATTRIBUTE-TYPE-VALIDATION-DATA MEDIUM-VALUES
```

along with the following meta-attribute clauses:

```
VALUE/RANGE = VALUE
DATA-VALUE = HARD-COPY, MICROFICHE, MICROFILM
```

which have the following meaning:

- o The first one of these clauses specifies that the attribute-type-validation-data meta-entity contains discrete values against which the checking is to occur. The legal values are given in the second clause. As a possible alternative, in the case where it is desired that code values are to be used for data entry to the dictionary, this second clause would have been stated as

```
DATA-VALUE = 1 "HARD-COPY"
              2 "MICROFICHE"
              3 "MICROFILM"
```

Whenever the CODE-VALUES-DEFAULT described in Section 2.2.5 is set to LITERALS, the transliterated values will appear in reports and in response to queries.

- o If the first clause had been specified as

```
VALUE/RANGE = RANGE
```

it would mean that the checking is to take place

against one or more ranges that are being specified. An example of such a case would be if it were desired to restrict attributes to fall in the range from 2 to 9, or in the range from 12 to 27.

- o The second clause would then specify the values that are permissible. Should it have been desired to specify the ranges mentioned above, then the required clause would be

DATA-RANGE = (2,9) (12,27)

We also need to point out that both the values and ranges thus specified can be altered. Suppose that at some future time a new medium for documentation is introduced called LASER, then the command

```
ALTER-META-ENTITY MEDIUM-VALUES  
DATA-VALUE FROM *null TO LASER
```

or alternately as

```
ALTER-META-ENTITY MEDIUM-VALUES  
DATA-VALUE FROM *null TO 4 "LASER"
```

would add LASER to the list of permissible values. (The symbol *null denotes here an implementor-defined symbol for a null value.) The command ALTER-META-ENTITY used here is specified in Section 5.1.4.

The next step that must be taken is to associate this attribute-type-validation-data meta-entity with the attribute-type MEDIUM. This is accomplished by the command

```
CREATE-META-RELATIONSHIP MEDIUM AND MEDIUM-VALUES
```

This command creates a meta-relationship which is an instance of the meta-relationship-type M-R-T(ATTRIBUTE-TYPE, ATTRIBUTE-TYPE-VALIDATION-DATA) of Section 3.1.2. Now one more step is required to achieve the overall objective. At this point we have the attribute-type MEDIUM associated with the attribute-type-validation-data MEDIUM-VALUES, but we still require the specification of a procedure that will carry out the actual checking. We then introduce another meta-entity-type called "attribute-type-valida-

tion-procedure" to indicate the checking that is to take place. The system-standard schema contains two such attribute-type-validation-procedures, one with the name VALUES, and the other one with the name RANGES. The core standard does not contain facilities for defining any other ones.

As these names imply, the attribute-type-validation-procedure called VALUES is used to check against lists of discrete values, i.e. attribute-type-validation-data for which VALUE/RANGE = VALUE has been specified, and the attribute-type-validation-procedure called RANGES is used for checking against ranges of values.

The next command that must then be issued is

CREATE-META-RELATIONSHIP MEDIUM AND VALUES

and the structure that we have created contains

- o the attribute-type MEDIUM which is associated with MEDIUM-VALUES (which contains the permissible attributes of MEDIUM)
- o the attribute-type MEDIUM which is associated with the attribute-type-validation-procedure VALUES (indicating that the checking is to take place against discrete values to be found in the associated attribute-type-validation-data).

Should it be desired by a user to find out what legal values exist for a given attribute-type, this can be achieved through the use of the META-LIST command of Section 5.2.3. For example, the command

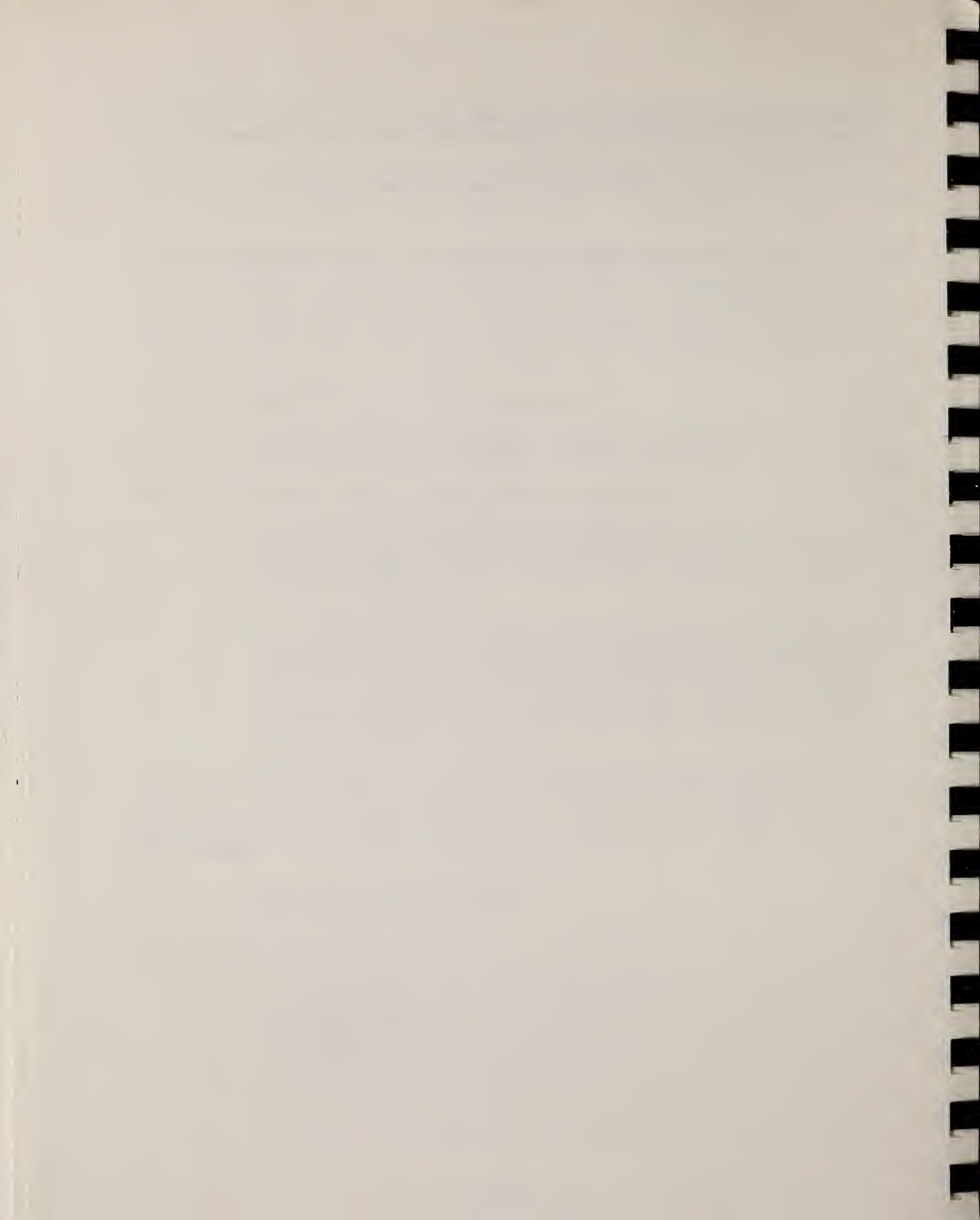
META-LIST MEDIUM-VALUES

would return the following:

- 1 "HARD-COPY"
- 2 "MICROFICHE"
- 3 "MICROFILM"
- 4 "LASER"

The preceding example does not cover all the meta-entity-types that exist in the schema. Others exist that are related to:

- o The statuses that exist in the dictionary and their structure.
- o The stages that are defined for dictionary entities.



U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 82-2619	2. Performing Organ. Report No.	3. Publication Date January 1983
4. TITLE AND SUBTITLE Functional Specifications for a Federal Information Processing System Data Dictionary System			
5. AUTHORS Editors Patricia A. Konig, Alan Goldfine, Judith J. Newton			
6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions) NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234 Alpha Omega Group, Inc. P.O. Drawer M Harvard, MA		7. Contract/Grant No. NB81SBCA0735	8. Type of Report & Period Covered
9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP) National Bureau of Standards Division 642, Tech., A265 Washington, D. C. 20234			
10. SUPPLEMENTARY NOTES <input type="checkbox"/> Document describes a computer program; SF-185, FIPS Software Summary, is attached.			
11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here) This interim report contains Functional Specifications for the basic functions that data dictionary software must perform to satisfy Federal agency requirements. The functionality specified will be incorporated into a planned Federal Information Processing Standard (FIPS) Data Dictionary System (DDS). The complete FIPS DDS also will contain additional specifications for such things as the user interface. Comments are being solicited from Federal agencies and suppliers of data dictionary software to determine any modifications that should be made to the Functional Specifications. Information about the effort to develop the planned FIPS DDS and a Management Overview of the Functional Specifications appear in Part I of this document. The Functional Specifications are in Part II.			
12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons) Computer program; data dictionary system; data inventory; data management; data standards; database; database management system; documentation; Federal Information Processing Standards Publication; requirements; software.			
13. AVAILABILITY <input checked="" type="checkbox"/> Unlimited <input type="checkbox"/> For Official Distribution. Do Not Release to NTIS <input type="checkbox"/> Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402. <input checked="" type="checkbox"/> Order From National Technical Information Service (NTIS), Springfield, VA. 22161			14. NO. OF PRINTED PAGES 410 15. Price

2



